VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Možnosti automatického generování kódu ve Visual Studiu 2008

Posibilities of Automatic Code Generation in Visual Studiu 2008

2011 Bc. Petr Hodovský

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Petr Hodovský**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Možnosti automatického generování kódu ve Visual Studiu 2008
Posibilities of Automatic Code Generation in Visual Studiu 2008

Zásady pro vypracování:

Diplomant se seznámí s problematikou automatického generování kódu pro Visual Studio 2008 (2010) se zaměřením na programovací jazyk C#, a to především technologie CodeSnippet, interní šablony Visual Studia, Domain-Specific Language, T4 Template a Guidance Package. Cílem práce je vytvořit ukázky použití těchto technologií. Diplomant zhodnotí možnosti a omezení použití jednotlivých technologií a porovná je mezi sebou.

Jednotlivé body práce jsou následující:
1. Seznámení se s obecnými principy generování kódu.
2. Seznámení a popsání technologii Code snippet.
3. Seznámení a popsání technologii „šablony Visual Studia".
4. Seznámení se s T4 Template, DSL a Guidance Package.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího diplomové práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Tomáš Kocyan**

Datum zadání: 20.11.2009

Datum odevzdání: 06.05.2011

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA * Fakulta elektrotechniky a informatiky * OSTRAVA

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení:

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

Datum: Podpis:

…… ……

Poděkování:

Na tomto místě bych rád poděkoval vedoucímu mé práce, Ing. Tomáši Kocyanovi, za konstruktivní připomínky. Kolegům z práce za inspiraci k napsání této práce. Dále pak Mgr. Blance Přikrylové a Bc. Veronice Forchové za korekturu celého textu. V neposlední řadě bych rád poděkoval svým rodičům, kteří mě podporovali po celou dobu mého studia.

Abstrakt

Práce se zabývá automatickým generováním kódu jako jednou z technik pro zjednodušení a zrychlení vývoje aplikací. Zaměřena je na vývojové prostředí Visual Studio 2008 a Visual Studio 2010. Jsou zde popsány technologie Code Snippets, Visual Studio Template, Text Template Transformation Toolkit, Domain Specific Language a Guidance Package. Každá z nich je popsána nezávisle na ostatních a jsou k ní uvedeny jednoduché příklady použití pro lepší pochopení. Práce se snaží o komplexní pohled na možnosti automatického generování kódu, protože touto problematikou i jednotlivými technologiemi se zabývá málo publikací a článků.

Klíčová slova: Code Snippets, VS Templates, C#, T4, Text Template Transformation Toolkit, DSL, Guidance Package

Abstract

This thesis is dealing with automatic code generation as a one of techniques for simplification and acceleration of application development. It is focused on Visual Studio 2008 and Visual Studio 2010 as development environment. Here are described these technologies Code Snippets, Visual Studio Template, Text Template Transformation Toolkit, Domain Specific Language and Guidance Package. Every-one of them is described independently on others and includes simple examples for better understanding. This thesis is attempted to be complex view on options of automatic code generation because there is a lack of publications and articles on this topic.

Key words: Code Snippets, VS Templates, C#, T4, Text Template Transformation Toolkit, DSL, Guidance Package

Seznam použitých zkratek a symbolů

IDE	– integrated development environment (integrované vývojové prostředí)
VS	– Visual Studio
VS 2008	– Visual Studio 2008
VS 2010	– Visual Studio 2010
SDK	– Software Development Kit
HTML	– Hyper Text Markup Language
ASP	– Active Server Pages
SQL	– Structured Query Language
XML	– Extensible Markup Language
VB	– Visual Basic
T4	– Text Template Transformation Toolkit
DSL	– Domain-Specific Language
GAC	– Global Assembly Cache
GP	– Guidance Package
GAT	– Guidance Automation Toolkit
GAX	– Guidance Automation Extensions

Obsah

Seznam obrázků

Nenalezena položka seznamu obrázků.

1 Úvod

V dnešní době je na vývoj softwaru kladeno několik požadavků, a to jak ze strany zákazníků, tak ze strany obchodníků prodávajících vyvíjený software. Mezi tyto požadavky patří zejména rychlost vývoje a rozšiřitelnost konečného produktu. Při vývoji se ale musí myslet na to, že zákazník v jeho průběhu bude měnit nebo alespoň upravovat své požadavky na vyvíjený produkt. Pro takovýto druh softwaru se osvědčilo používat modulárního vývoje, aby bylo možno rozložit vytváření produktu mezi více vývojářů a případně i mezi více vývojářských týmů.

Modulární řešení řeší hned několik problémů, které byly zmíněny. Zrychlení vývoje je nesporné, neboť to, co by programoval jeden člověk měsíce, tým zvládne v mnohem kratší době. Pokud je produkt rozdělen na moduly, mělo by být jejich načítání dynamické, aby byla zajištěna i případná rozšiřitelnost. Modularita sebou však nese určité problémy, které si někteří programátoři nebo i vedoucí vývojových týmů neuvědomují. Tímto problémem je například neustálé opisování stejných úseků kódu, při němž dochází nejen ke značné chybovosti, ale i ke ztrátě času. Místo toho, aby se vývojář soustředil na vlastní logiku požadovaného modulu, tak řeší, proč mu kód, který již tolikrát napsal, zrovna teď, když má tak málo času a potřebuje mít práci hotovou, nefunguje. V tomto okamžiku vzniká napětí v týmu a současně, pokud se nepodaří splnit termín, i mezi zákazníkem a dodavatelem softwaru. Nastává otázka, jak vytvářet modulární aplikaci a přitom neopisovat stále stejný kód. Řešením je použít automatické generátory kódu.

Vytváří-li se aplikace pomocí Model-Driven Development v n-vrstvé architektuře, naráží se zde na problém, jak udržet konzistentní model při změnách požadavků zákazníka nebo při rozšiřování funkčnosti aplikace. Například máme-li již existující databázovou tabulku implementačně zahrnutou v našem řešení a chceme ji rozšířit o nějaký sloupec. Zpravidla to znamená rozšířit databázovou tabulku v programu objekt, který tuto tabulku reprezentuje, dále musíme změnit načítání tohoto sloupce do objektu. Případně ještě ošetřit další věci, jako je zobrazení v dialogu a ošetření uživatelských vstupů. I zde se dá velice dobře využít automatického generování kódu.

“If you’re going to do something twice or more, manually, in your company, generate it.”

Scott Hanselman

1.1 Automatické generování kódu

Automatické generování není novinkou posledních několika let, ale využívá se již dlouhou dobu, aniž si to třeba uvědomujeme. Například při odesílání e-mailu se nám celá zpráva obalí, aby byly její jednotlivé části, jako je hlavička a tělo, srozumitelné. A to jak pro e-mailový server, tak i e-mailového klienta, kam má být zpráva doručena, a aby mohla být správně interpretována. Ke generování dochází i při překladu z vyššího programovacího jazyka do strojového kódu. Proč tedy nevyužít automatické generování i v průběhu vývoje softwaru? I tady jistě každý používá automatické generování, aniž by si to uvědomil nebo přemýšlel nad tím, že by se technika, kterou používá ve svém vývojovém prostředí (IDE), dala rozšířit, případně ještě více přizpůsobit vlastním požadavkům.

Automatické generování kódu je použitelné ve všech fázích vývoje, jak již bylo nastíněno. Nyní si jen letmo představíme některé technologie. Při vytváření nových modulů využijeme například Visual Studio Templates (viz. kapitola 3) pro generování nových projektů do Visual Studia (VS). Během psaní vlastního kódu programu velmi snadno a pohodlně můžeme využít Code Snippets (viz. kapitola 2), které jsou vhodné pro generování krátkých i delších kusů kódu, jež se neustále s jistou obměnou opakují. Pro jednodušší a konzistentní vývoj n-vrstvé architektury lze použít modelování pomocí Domain Specific Language (DSL) (viz. kapitola 5) a následného generování kódu z tohoto DSL T4 šablonami (viz. kapitola 4).

Automatické generování má samozřejmě nejen svoje velké výhody, ale i nevýhody. Teď si některé z výhod i nevýhod zkusíme vyjmenovat.

Začněme výhodami, ty chce slyšet každý jako první, protože každý chce vědět, co mu nová technologie přináší.

a. sjednocení vzhledu výsledného textu a vetší čitelnost kódu pro ostatní programátory v týmu,
b. konzistentní kvalita kódu,
c. pokud má dojít ke změně, mění se jen generátor,
d. zrychlí vývoj celé aplikace;

Nyní ještě zmíníme nevýhody, aby byl seznam úplný. Také bychom měli být obeznámeni s oběma stranami mince.

a. generátor musí být nejprve napsán a odladěn,
b. tvorba složitějších generátorů vyžaduje značné zkušenosti jak z hlediska programování v cílovém jazyce, tak i v prostředí nebo jazyce, ve kterém generátor vytváříme,

c. automatické generování nelze použít vždy a všude,
d. vždy je třeba nějaká část programu napsat ručně;

2 Code Snippets

Jednou z prvních věcí, kterou by se měl dnešní vývojář naučit, je neopakovat sám sebe. Co je tím myšleno? Každý programátor se jistě ocitl v situaci, že dlouhé minuty přemýšlel, proč mu část programu nefunguje, a nakonec zjistil, že má špatnou syntaxi cyklu nebo mu chybí nějaká závorka. Cílem programátora není psát stále stejný kód opakovaně a přemýšlet nad syntaxí cyklu, větvení atd. Jeho úkolem je psát program a vymýšlet postup řešení zadaného problému. Jednou z nejjednodušších technik, jak tomuto opisování sám sebe zamezit, je Code Snippets (nebo-li útržky kódu), které jsou přímo integrované ve VS.

Princip Code Snippets si popíšeme ve třech jednoduchých krocích, na kterých bude zřejmé, jak tato technika funguje. Nejprve se napíše klíčové slovo, jež je zvoleno jako zkrácené jméno Code Snippet. Po napsání tohoto slova nebo lépe řečeno písmen se zobrazí nápověda (viz. Obrázek 1). Jakmile stiskneme klávesu „Tab“ automaticky se vypíše text (viz. Obrázek 2), kde jsou vidět zvýrazněná slova, a ta je možno během vkládání editovat. Při upravení jednoho výskytu slova se upraví i všechny ostatní výskyty (viz. Obrázek 3). Když je kód takový, jaký potřebujeme, stiskneme klávesu „Enter“ a tím ukončíme editaci. Od této chvíle se již vložený kód chová jako jakýkoliv jiný zdrojový text. Vše si ještě podrobněji popíšeme v kapitole 2.2.

Obrázek 1: Nápověda pro vložení Code Snippet

```
for (int i = 0; i < UPPER; i++)
{

}
```

Obrázek 2: Vložený Code Snippet s automaticky předvyplněnými hodnotami

```
for (int index = 0; index < UPPER; index++)
{

}
```

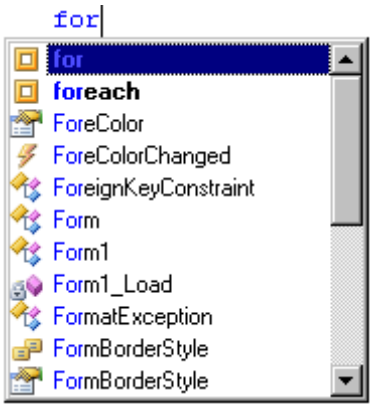
Obrázek 3: Code Snippet během editace doplňuje všechny výskyty proměnné

2.1 Obecné informace

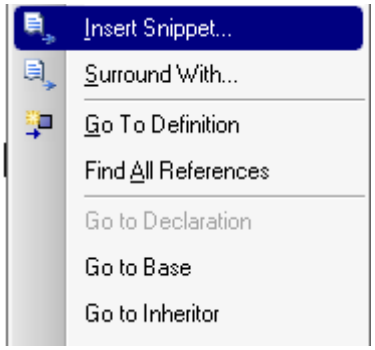
Ve Visual Studiu od verze 2005 je podporováno vkládání tzv. Code Snippets, což umožňuje rychlé vložení často se opakujícího kusu kódu. Pomocí Code Snippets je možno ve VS 2008 generovat zdrojový kód pro jazyky C#, Visual Basic (VB), J# a značkovací jazyk XML. VS 2010 již podporuje i jazyk HTML a technologii ASP.NET, ale i skriptovací jazyk T – SQL.

2.2 Použití

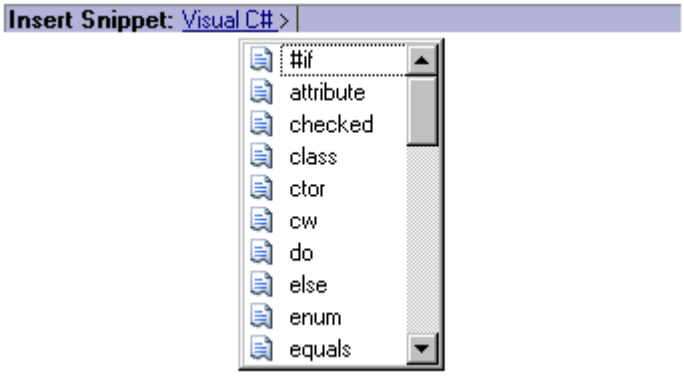
Code Snippets se používá především k automatickému generování menších částí kódu, ale touto technologií lze generovat jakkoli dlouhý kód, ovšem vše jen v rámci jednoho zdrojového souboru. Vložení Code Snippets je možno provést dvěma základními způsoby, vložením přes IntelliSense (viz. Obrázek 4) nebo přes kontextové menu (viz. Obrázek 5 a Obrázek 6). IntelliSense je funkce automatického dokončování slov spojená s nápovědou.



Obrázek 4: IntelliSense pro CodeSnippet



Obrázek 5: Kontext menu s vybranou položkou pro vložení Code Snippetu



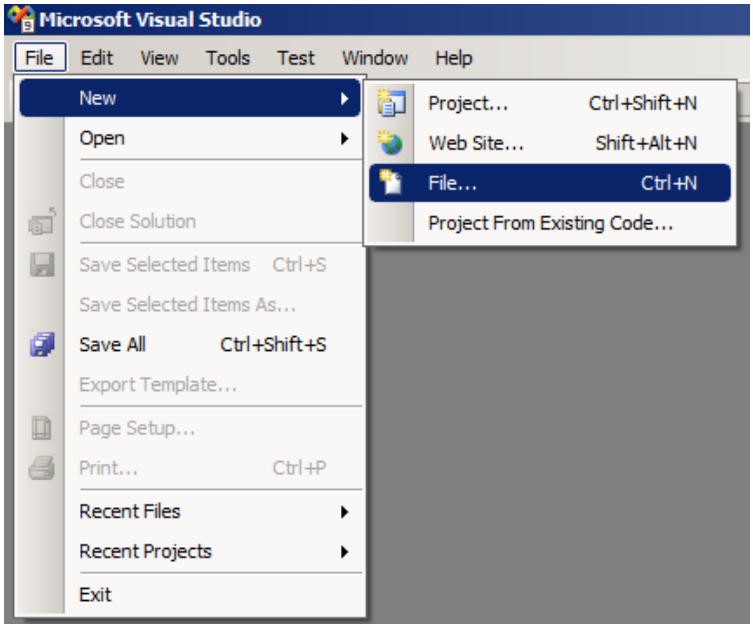
Obrázek 6: Seznam použitelných Code Snippets

Pokud má Code Snippet zkratku, tak se nám zobrazí v InteliSense a po jeho vybrání se vloží příslušný kód. Nezobrazí-li se nám InteliSense, ale víme, že taková zkratka existuje, stačí stisknout klávesu „Tab“ a kód se také vloží. V takto vloženém kódu lze editovat nadefinované části a mezi jednotlivými položkami se přepíná pomocí klávesy „Tab“.

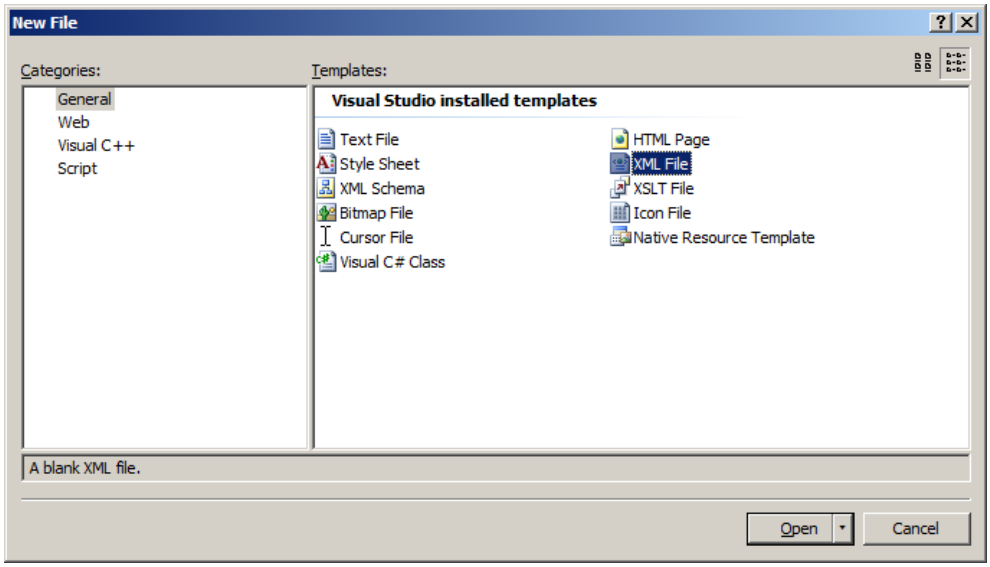
Vyvolání z kontextového menu se provede označením položky „Insert Snippet…“ nebo „Surround With…“, poté se zobrazí menu (viz. Obrázek 5). V tomto menu se můžeme pohybovat šipkami a požadovaný útržek kódu vložit klávesou „Enter“. Poté je postup obdobný, jako jsme si již popsali při vkládaní přes InteliSense.

2.3 Vytvoření nového Code Snippet

Nyní si přiblížíme vytvoření nového Code Snippet. Otevřeme si VS a v menu vybereme „File > New > File“ (viz. Obrázek 7) nebo klávesovou zkratkou Ctrl+N. V dialogu „New File“ vybereme kategorii „General“ a v ní položku „XML File“.

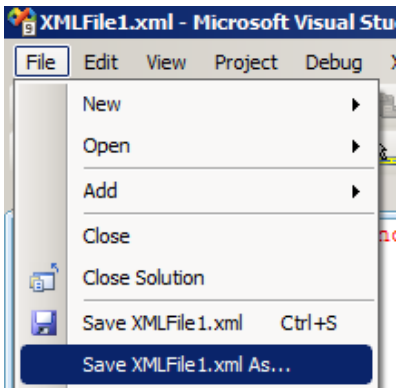


Obrázek 7: Menu pro otevření nového souboru

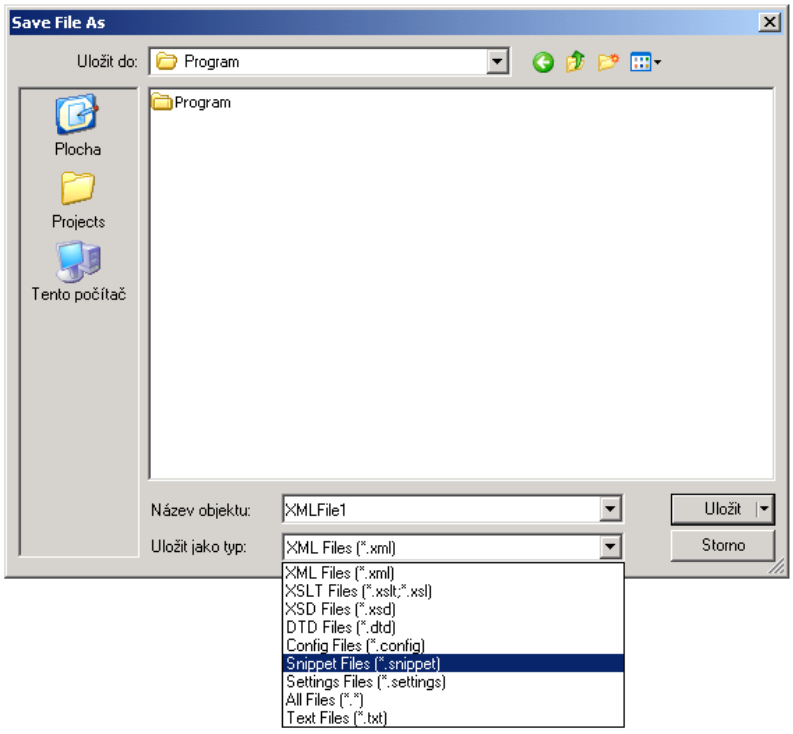


Obrázek 8: „New File“ dialog

Soubor dáme uložit přes menu „File > Save As…“ (viz. Obrázek 9) a uložíme jako *.snippet (viz. Obrázek 10). Po uložení souboru vložíme kostru Code Snippet pomocí útržku Code Snippet, klikneme pravým tlačítkem myši a v menu zvolíme „InsertSnippet...“ a z nabídky vybereme „Snippets“ a následně položku s názvem „Snippet“.



Obrázek 9: Menu pro uložení souboru



Obrázek 10: „Save File“ dialog, ukládání XML jako *.snippet

Tím se nám vloží celá kostra Code Snippet s editovatelnými položkami (viz. Obrázek 11), po jejich vyplnění stiskneme klávesu „Enter“. Ještě změníme jazyk v atributu `Language` v elementu `Code` na `CSharp` (pokud chceme vytvářet Code Snippet pro C#) a nakonec mezi „`<![CDATA[`“ a „`]]>`“ vepíšeme náš kód. Více se o jednotlivých elementech a jejich atributech dovíme v kapitole 2.5.

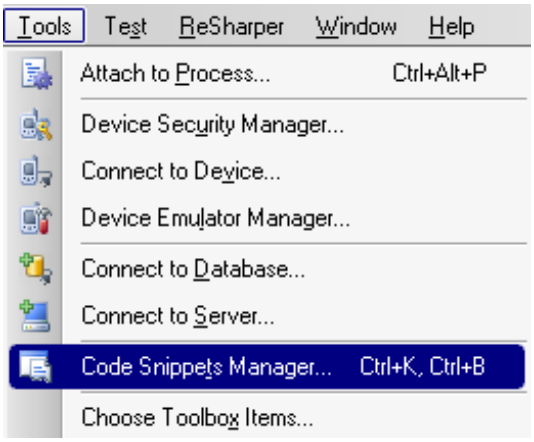
```
<?xml version="1.0" encoding="utf-8"?>
<CodeSnippet Format="1.0.0"
xmlns="http://schemas.microsoft.com/VisualStudio/2005/CodeSnippet">
  <Header>
    <Title>title</Title>
    <Author>author</Author>
    <Shortcut>shortcut</Shortcut>
    <Description>description</Description>
    <SnippetTypes>
      <SnippetType>SurroundsWith</SnippetType>
      <SnippetType>Expansion</SnippetType>
    </SnippetTypes>
  </Header>
  <Snippet>
    <Declarations>
      <Literal>
        <ID>name</ID>
        <Default>value</Default>
      </Literal>
    </Declarations>
    <Code Language="XML">
      <![CDATA[<test>
      <name>$name$</name>
      $selected$ $end$</test>]]>
    </Code>
  </Snippet>
</CodeSnippet>
```

Obrázek 11: Vložený Code Snippet

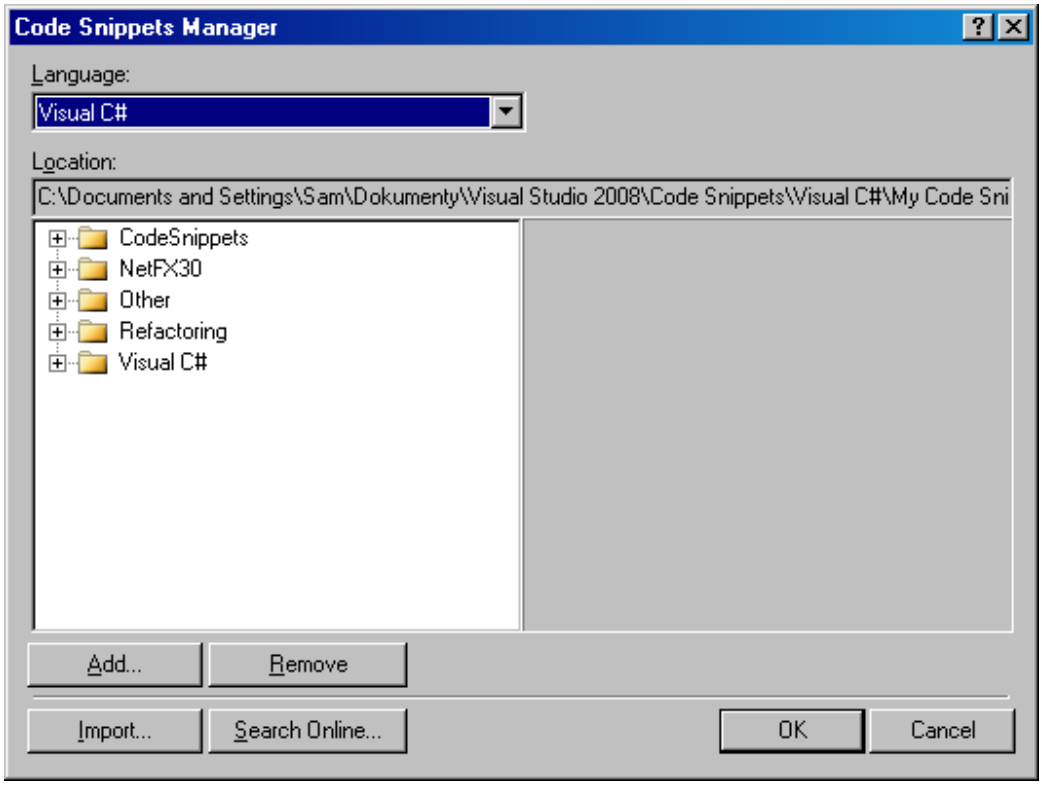
2.4 Vložení existujícího Code Snippetu do Visual Studia

Pro přidání Code Snippet do VS je zde dialog „Code Snippets Manager“. Jak dialog vyvolat a jak v něm pracovat, si nyní popíšeme.

Dialog vyvoláme z nabídky „Tools > Code Snippets Manager“ (Ctrl+K, Ctrl+B). Nyní máme dvě možnosti, jak vložit vytvořený útržek, „Import“ nebo „Add“. Při importu vložíme jeden konkrétní útržek. Při vkládání pomocí tlačítka „Add“ se vybírá složka na disku, ze které se mají útržky přidávat do VS.



Obrázek 12: Nabídka pro výběr Code Snippets Manageru



Obrázek 13: Dialog Code Snippets Manager

2.5 Popis XML elementů

V následujících řádcích si stručně popíšeme jednotlivé značky (tagy nebo také elementy) a jejich případné atributy s výčtem jejich použití především pro jazyk C#. Schéma Code Snippets jak grafické, tak textové je v přílohách (viz. A).

2.5.1 Element „CodeSnippets“

Kořenový element celého Code Snippets XML souboru. Element `CodeSnippets` může obsahovat několik podelementů `CodeSnippet`. Pro aktivaci InteliSense ve VS je nutno definovat atribut `xmlns`.

2.5.2 Element „CodeSnippet“

Obsahuje již konkrétní definici jednoho Code Snippet, který se bude vkládat. Tento element má atribut `Format`, který určuje verzi Code Snippet. V době psaní textu nabývá pouze hodnoty `1.0.0`.

Potomci elementu `CodeSnippet` jsou elementy `Header` a `Snippet`, které je nutno uvést v tomto pořadí.

2.5.3 Element „Header“

Element `Header`, jak již název napovídá, definuje hlavičku jednoho Code Snippet. Seznam obecných informací uvedených v jednotlivých podelementech si přiblížíme krátkým popisem:

a. `Author` – jméno autora,
b. `Description` – textový popis útržku kódu,
c. `Keywords` – kontejner pro XML elementy `Keyword`,
d. `Shortcut` – zkratka, použitá při vkládání do kódu v InteliSense ve VS,
e. `Title` – jméno pro Code Snippet;

2.5.4 Element „Snippet“

Druhým potomkem elementu `CodeSnippet` je element `Snippet`, kde se realizuje vlastní definice Code Snippet. Tento element nemá žádný atribut, ale obsahuje dva pod elementy `Code` a `Declarations`.

2.5.5 Element „Code“

Do elementu `Code` se umisťuje vlastní kód, který se bude vkládat při spuštění Code Snippet. Element `Code` má atribut `Language`, kde se píše jméno cílového jazyka, pro který je Code Snippet určen. V současné době jsou ve VS 2008 podporovány jazyky C#, J#, Visual Basic a XML. Ve VS 2010 ještě přibývá podpora HTML, ASP.NET, JScript a T-SQL. Do elementu `Code` je možno napsat libovolný text reprezentující určitý kód a proměnné, které lze následně editovat. Proměnná se zapisuje mezi dva znaky ‘$’ (dolar) a v dalších elementech se musí definovat a určit její typ. Kromě uživatelských proměnných existují v Code Snippets ještě speciální proměnné a proměnné s funkcí.

Speciální proměnné jsou `$end$` a `$selected$`. Proměnná `$end$` slouží k umístění kurzoru do kódu po dokončení editace Code Snippets. Pokud se tato proměnná neumístí do kódu, kurzor je automaticky umístěn na konec vkládaného kódu. Proměnná `$selected$` slouží k vložení označeného textu ve zdrojovém kódu do těla Code Snippets.

Proměnné s funkcí lze libovolně pojmenovat, ale přiřazuje se jim jedna ze tří funkcí `SimpleTypeName()`, `ClassName()` a `GenerateSwitchCases()`. Po přiřazení funkce je již nelze editovat. O těchto funkcích bude zmínka v kapitole 2.5.9.

2.5.6 Element „Declarations“

Druhým potomkem elementu `Snippet` je element `Declarations`. Tento element je nepovinný a obsahuje specifikaci proměnných, které tvoří část Code Snippet. Element `Declaration` neobsahuje žádné atributy.

2.5.7 Element „Literal“

Element `Literal` je podelementem elementu `Declarations` a používá se pro specifikaci uživatelských proměnných obsahujících řetězce, číselné hodnoty a proměnné.

2.5.8 Element „Object“

Element `Object` lze použít pro deklaraci objektů v kódové šabloně. Jak element `Object`, tak i element `Literal` mají stejný atribut `Editable`, určující zda je proměnná citovatelná po vložení Code Snippet, i kolekci podelementů.

Podelemety elementů `Literal` a `Object`:

a. `ID` – identifikátor elementu,

b. `Default` – výchozí (implicitní) hodnota,
c. `Function` – nastavení funkce, viz. níže,
d. `ToolTip` – text popisku zobrazené uživatelům,
e. `Type` – typ objektu;

2.5.9 Element „Function“

Do tohoto elementu se vpisuje funkce, která se vykoná při vkládání kódu. Pro použit tohoto elementu se nastavení hodnota atributu `Editable` na `false`. Existují tři funkce, které lze použít `ClassName()`, `GenerateSwitchCases(EnumerationLiteral)` a `SimpleTypeName(TypeName)`. Jednotlivé funkce si nyní popíšeme:

a. `SimpleTypeName(TypeName)` – redukuje `TypeName` na jeho nejkratší zápis, který je možný v daném kontextu kódu. Např. při zápisu v Code Snippet `<Function>SimpleTypeName(global::System.Console)</Function>` a při již nareferencovaném balíku `System`, se vloží pouze `Console`, pokud reference chybí vloží se `System.Console`.
b. `ClassName()` – vrací jméno třídy, do které je Code Snippet vkládán.
c. `GenerateSwitchCases(EnumerationLiteral)` – tato funkce se využívá při použití `switch` příkazu v Code Snippetu. Pokud bude `EnumerationLiteral` nějaký výčtový typ např. `RowState`, budou vygenerovány všechny přípustné hodnoty tohoto typu (viz. Obrázek 14 a Obrázek 15).

```
DataRow radek = new DataRow();
switch (radek.RowState)
{
    case DataRowState.Added:
        break;
    case DataRowState.Deleted:
        break;
    case DataRowState.Detached:
        break;
    case DataRowState.Modified:
        break;
    case DataRowState.Unchanged:
        break;
    default:
        break;
}
```

Obrázek 14: Kód vygenerovaný pro výčtový typ RowState.

```
char c = new char();
switch (c)
{
    default:
        break;
}
```

Obrázek 15: Kód vygenerovaný pro primitivní datový typ.

2.6 Komunitní nástroje

SnippetEditor 2.1

http://snippeteditor.codeplex.com/

CodeSnippetsEditor

http://codesnippeteditor.codeplex.com/

ExportAsCodeSnippet

http://www.codeplex.com/ExportAsCodeSnippet

Snipp Dogg v1.5

http://code.msdn.microsoft.com/SnippDogg

2.7 Shrnutí

Nebyly zde popsány všechny elementy a jejich atributy, protože některé tyto elementy a atributy nejsou pro psaní Code Snippets pro jazyk C# použitelné, nebo nejsou určené pro tento jazyk.

CodeSnippets.snippet

```
1  <?xml version="1.0" encoding="utf-8" ?>
2  <CodeSnippets  xmlns="http://schemas.microsoft.com/VisualStudio/2005/CodeSnippet">
3    <CodeSnippet Format="1.0.0">
4      <Header>
5        <Title>for</Title>
6        <Shortcut>for</Shortcut>
7        <Description>Code snippet for 'for' loop</Description>
8        <Author>Microsoft Corporation</Author>
9        <SnippetTypes>
10         <SnippetType>Expansion</SnippetType>
11         <SnippetType>SurroundsWith</SnippetType>
12       </SnippetTypes>
13     </Header>
14     <Snippet>
15       <Declarations>
16         <Literal>
17           <ID>index</ID>
18           <Default>i</Default>
19           <ToolTip>Index</ToolTip>
20         </Literal>
21         <Literal>
22           <ID>max</ID>
23           <Default>length</Default>
24           <ToolTip>Max length</ToolTip>
25         </Literal>
26       </Declarations>
27       <Code Language="csharp">
28         <![CDATA[for (int $index$ = 0; $index$ < $max$; $index$++)
29          {
30            $selected$ $end$
31          }]]>
32       </Code>
33     </Snippet>
34   </CodeSnippet>
35 </CodeSnippets>
```

100 %

Obrázek 16: Použití Code Snippets pro for cyklus

3 Templates

Hlavní myšlenkou Templates pro VS je sjednotit a zrychlit zakládání nových projektů při vývoji a pokud používají stejný základ, tak zautomatizovat a zrychlit jejich přípravu, vytváření a minimalizovat chyby pramenící z kopírování existujících kódů.

Templates jsou šablony, které popisují, jak se má založit nový soubor, projekt nebo skupina projektů. VS Templates používáme kdykoli, kdy zakládáme nový projekt ve Visual Studiu nebo do existujícího projektu vytváříme nový soubor.

3.1 Obecné informace

Stejně jako Code Snippets je i tvorba šablon poprvé použita ve VS 2005. Pro popis šablony je použit XML soubor popisující její strukturu. Rozlišujeme tři druhy šablon, „Project template“, „Item template“ a „ProjectGroup template“. Všechny varianty budou popsány níže. Existují dva druhy použití šablony základní a rozšířené o průvodce (tzv. wizarda), který slouží pro dosazení hodnot za proměnné v kódu.

3.2 Použití

Základem šablony je zdrojový soubor pro „Item template“ nebo zdrojový projekt pro “Project template“ a soubor s příponou .vstemplate. V případě „ProjectGroup template“ je to několik samostatných šablon. To vše zabaleno do jednoho archivu ve formátu zip. Tento archiv je připraven pro použití jako šablona. Zbývá jen nakopírovat archiv do složky „[*Dokumenty*]\[*Verze VS*]\Templates\ProjectTemplates“ v případě „Project template“ a v případě „Item template“ se kopíruje archiv do složky „[*Dokumenty*]\[*Verze VS*] \Templates\ItemTemplates“. Pokud je vše správně vytvořeno, pak uvidíme tento náš nový projekt v nabídce „File > New > Project…“ (nebo „File > New > File…“). Je-li něco v nepořádku, novou šablonu neuvidíme nebo nám nepůjde použít.

3.3 Popis šablony

V následujících řádcích budou popsány stručně nejpoužívanější tagy, které lze využít při tvorbě šablon a případné atributy těchto tagů se zaměřením na použití především pro jazyk C# s ohledem na „WinForm“ projekty. Existují speciální tagy a atributy některých tagů,

které jsou pro webové projekty. Těmto tagům a atributům se ale věnovat nebudeme. Schémata šablon pro „Item template“, „Project template“ i „ProjectGroup template“ jak grafické, tak textové jsou v přílohách (viz. B).

3.3.1 Element „VSTemplate“

Element `VSTemplate` je kořenovým elementem pro celou šablonu. Obsahuje tři atributy:

a. `Type` – typ šablony (Project nebo Item).
b. `Version` – verze šablony (v době psaní textu byla pro verze VS 2005, VS 2008 i VS 2010 hodnota 2.0.0).
c. `xmlns` – schéma šablony. Tento atribut je nepovinný jako v předešlé technologii a slouží pouze pro podporu InteliSense ve Visual Studiu.

Element musí mít dva povinné podelementy `TemplateData`, `TemplateContent` a může mít dva volitelné podelementy `WizardData`, `WizardExtension`.

3.3.2 Element „TemplateData“

Element `TemplateData` musí být před elementem `TemplateContent`, poskytuje obecné informace o šabloně, jako je její jméno, popis, ikona atd.

Podelementy elementu `TemplateData` (uvedeme zde jen některé, ty důležitější. Kompletní přehled naleznete v příloze):

a. `DefaultName` – implicitní název, který se objeví v „New Project“ nebo v dialogu „Add New Item“.
b. `Description` – řetězcová hodnotu, která se bude zobrazovat jako popis šablony. Má dva volitelné atributy: `ID` a `package`. `ID` je zdrojový identifikátor pro `VS`, a `Package` je GUID, který se odkazuje na VS balíček s `ID`.
c. `EnableEditOfLocationField` – volba povolující změnu pole umístění projektu
d. `Hidden` – viditelnost šablony v „New Project“ nebo „Add New Item“ dialogu. V tomto případě není potřeb nastavovat jiné podelementy `TemplateData`.
e. `Icon` – jméno souboru s příponou .ico (ikonou). Tato ikona se bude zobrazovat v dialogu pro výběr šablony. Má dva volitelné atributy: `ID` a `package`. Obsah je stejný jako u elementu `Description`.

f. `Name` – jméno objevující se v „New Project“ nebo „Add New Item“ dialogu. Má dva volitelné atributy: `ID` a `package`. Obsah je stejný jako u elementu `Description`.
g. `ProjectType` – nastaví, kde se zobrazí v dialogu „New Project“ nebo „Add New Item“ dialogu. Toto je povinný podelement `TemplateData` a má čtyři možné hodnoty: CSharp, VisualBasic, JSharp, a Web.
h. `ProvideDefaultName` – určuje, zda implicitní název má být zobrazen, pokud je `false`, tak v položce `Name` bude „<Enter_name>“.
i. `RequiredFrameworkVersion` – hodnota udává minimální verzi .NET FRAMEWORKu pro použití šablony. Možné hodnoty jsou 2.0,3.0, 3.5 a 4.0.
j. `SortOrder` – upřesňuje pořadí řazení šablony v „New Project“ dialogu. Pokud není element uveden, projekty se řadí podle abecedy.

3.3.3 Element „TemplateContent“

Druhým povinným podelementem elementu `VSTemplate` je element `TemplateContent`, který definuje strukturu souborů a jejich jmen v šabloně. Tento prvek obsahuje právě jeden z podelementů `Project`, `ProjectCollection`, `ProjectItem`, `Reference` a volitelný element `CustomParameters`, význam tohoto elementu si vysvětlíme až při popisu tvorby průvodce.

3.3.4 Element „Project“

Element udržuje seznam souborů a složek, které mají být přidány do projektu. Obsahuje tři atributy a dva volitelné podelementy `Folder` a `ProjectItem`.
Atributy specifikují:

a. `File` – tento povinný atribut udává jméno projektového souboru v šabloně.
b. `ReplaceParameters` – logická hodnota určující zda projektový soubor má hodnoty parametrů, které je se musí nahradit při tvorbě projektu ze šablony.
c. `TargetFileName` – jméno projektového souboru při vytváření projektu.

3.3.5 Element „Folder“

Element definuje složku vkládanou do projektu. Povinný atribut `Name`, určuje jméno složky. Pokud je složka generována ze šablony její jméno určuje nepovinný atribut `TargetFolderName`.

3.3.6 Element „ProjectItem“

`ProjectItem` specifikuje cestu k souboru vkládanému do projektu v rámci šablony. Element se může nacházet v Project nebo Item šabloně. Jeho základními atributy jsou nám již známé:

a. `TargetFileName`,
b. `ReplaceParameters`;

3.3.7 Element „ProjectCollection“

Tento element je dalším podelementem `TemplateContent`. Používá se při vytváření hierarchické struktury pro vkládání projektů do multiprojektové šablony. Element neobsahuje žádné atributy, ale má dva volitelné podelementy:

a. `ProjectTemplateLink` – odkazuje se na projekt v multiprojektové šabloně. Atribut `ProjectName` uvádí cestu k další šabloně.
b. `SolutionFolder` – specifikuje složku v multiprojektové šabloně pro skupinu projektů. Atribut `Name` obsahuje její jméno.

Elementy `SolutionFolder` může ve svém těle obsahovat další elementy `ProjectTemplateLink` a `SolutionFolder` tak, aby definovaly hierarchii vytvářených souborů.

3.3.8 Element „References“

Tento element ve svém těle obsahuje alespoň jeden element `Reference`. Element `Reference` je kontejner pro další element `Assembly`, který obsahuje jméno assembly pro přidání do projektu. Element `References` může být přidán pouze do šablony typu `Item`.

3.3.9 Element „WizardData“

Element `WizardData` udržuje nějaké XML data, které je možno použít pro konfiguraci průvodce. Nad touto části šablony se neprovádějí validace.

3.3.10 Element „WizardExtension“

Tento element se stará o správné připojení průvodce k šabloně. Element nemá žádné atributy, pouze dva povinné podelementy:

a. `Assembly` – knihovna implementující průvodce.
b. `FullClassName` – konkrétní třída, kde je průvodce naimplementovaný. Tato třída je součástí assembly specifikované v elementu `Assembly`.

3.3.11 Proměnné

Proměnné se zapisují mezi znaky '$'(dolar) a je možné je pomocí průvodce nahradit konkrétní hodnotou nebo je možné v souboru *.vstemplate definovat jejich konkrétní hodnoty. Popis níže uvedených „systémových" proměnných, které jsou přímo zabudovány ve VS, je možno nalézt v příloze.

a. clrversion
b. GUID [1-10]
c. itemname
d. machinename
e. projectname
f. registeredorganization
g. rootnamespace
h. safeitemname
i. safeprojectname
j. time
k. userdomain
l. username
m. year

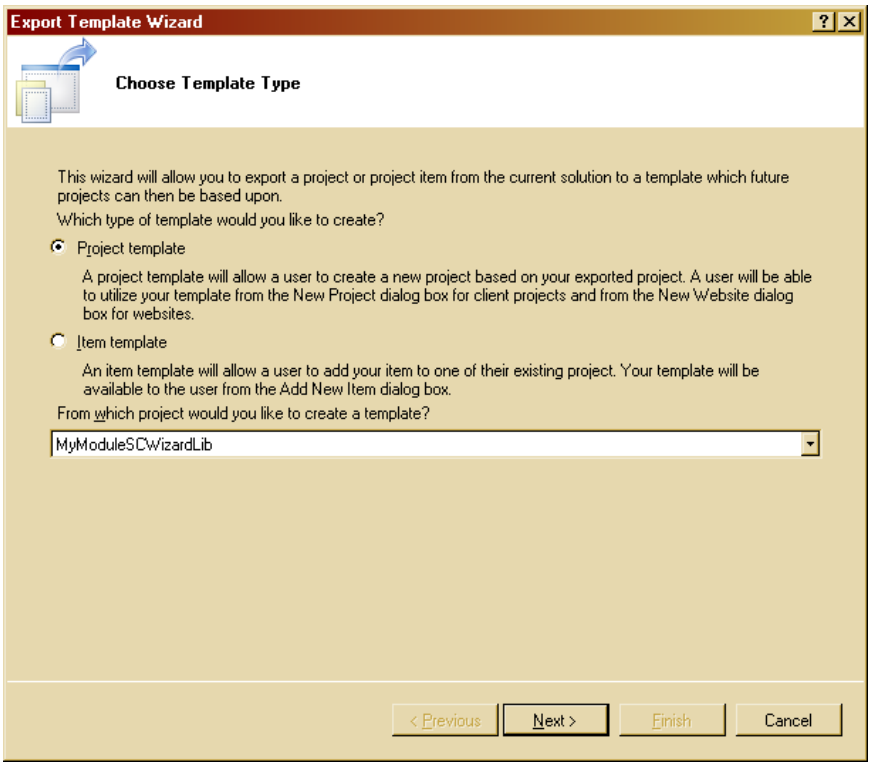
3.4 Vytvoření Template

Vytvoření jak „Project template", tak „Item template" z existujícího projektu je velmi jednoduché, avšak tato jednoduchost je vykoupena mírou variability vygenerované šablony. Variabilitu je možné zvýšit napsáním si vlastního průvodce (tzv. Wizarda). Popis vytvoření wizarda si popíšeme v kapitole 3.5. Nyní si popíšeme automatické vytvoření šablony, ruční vytvoření nemá velký význam popisovat, protože se příliš nepoužívá.

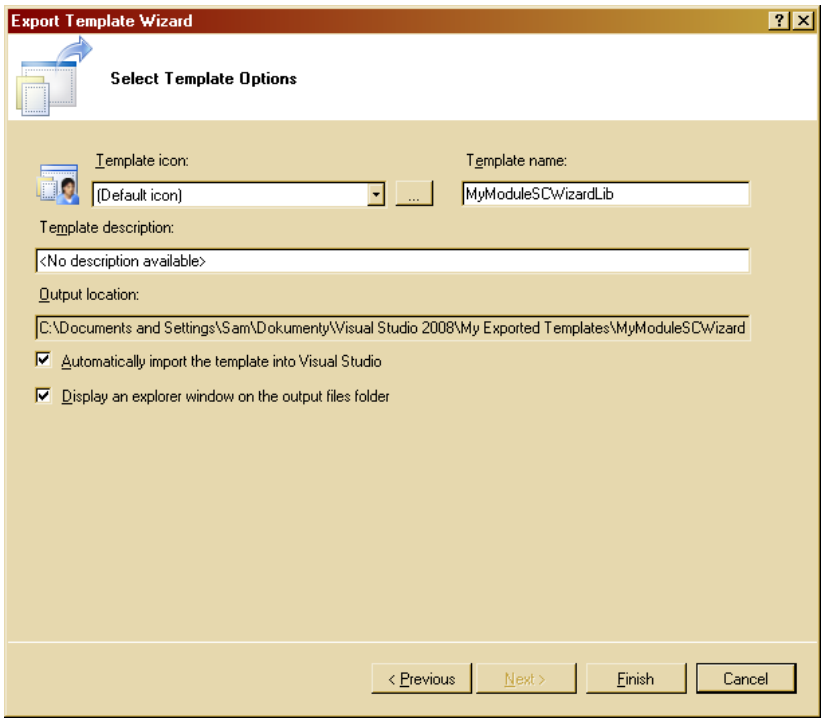
3.4.1 Automatické vytvoření

Předpokládejme, že již máme vytvořen projekt, ze kterého chceme udělat šablonu. V horním menu zvolíme „File > Export Template…", objeví se nám dialog (viz. Obrázek 17), ve kterém si můžeme zvolit, zda chceme exportovat „Project template" nebo „Item template" a pod tím je volba projektu, z kterého se budou exportovat data, pokud je v Solution více projektů. Nyní si vybereme „Project Temlate" a stiskneme „Next". K popisu postupu vygenerování „Item template" se vrátíme později. V následujícím dialogu (viz. Obrázek 18) je

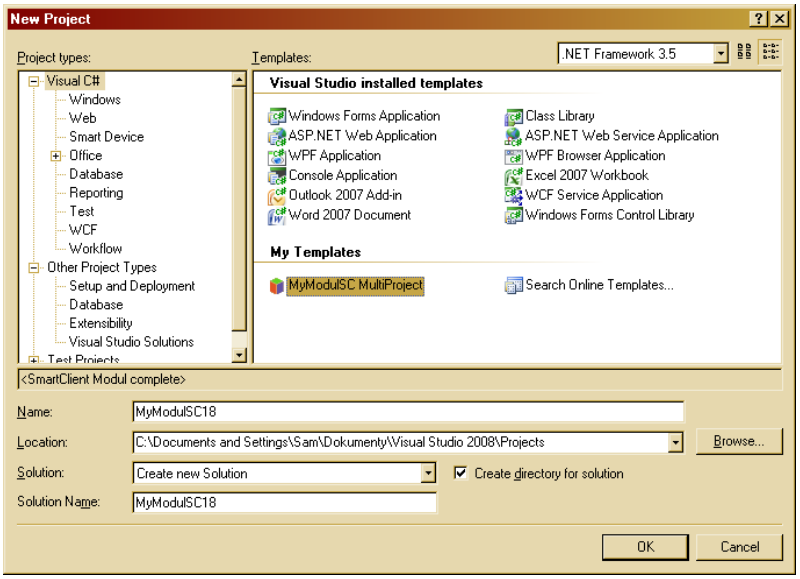
možné si zvolit ikonu pro šablonu. Vedle se píše jméno šablony, které bude vidět v dialogu pro vložení nového projektu, jak je vidět v dialogu (viz. Obrázek 19). Další textové pole slouží pro stručný popis šablony a následuje cesta, kam se exportovaná šablona uloží. Jako poslední jsou zde volby, zda se má šablona vložit do šablon VS a jestli se má zobrazit adresář s uloženou šablonou. Nakonec již zbývá jen stisknout tlačítko „Finish“.



Obrázek 17: Export Template dialog (volba „Project template“ nebo „Item template“)



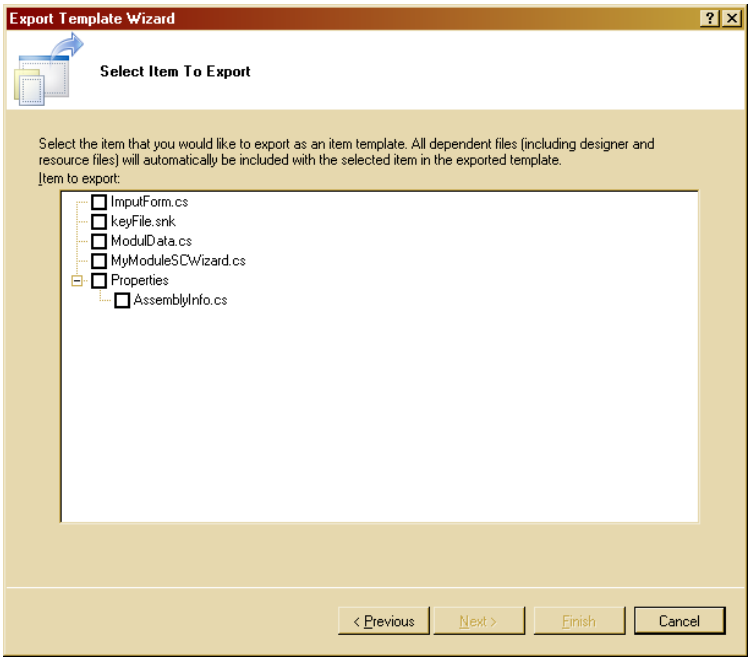
Obrázek 18: Export template (obecné informace o šabloně)



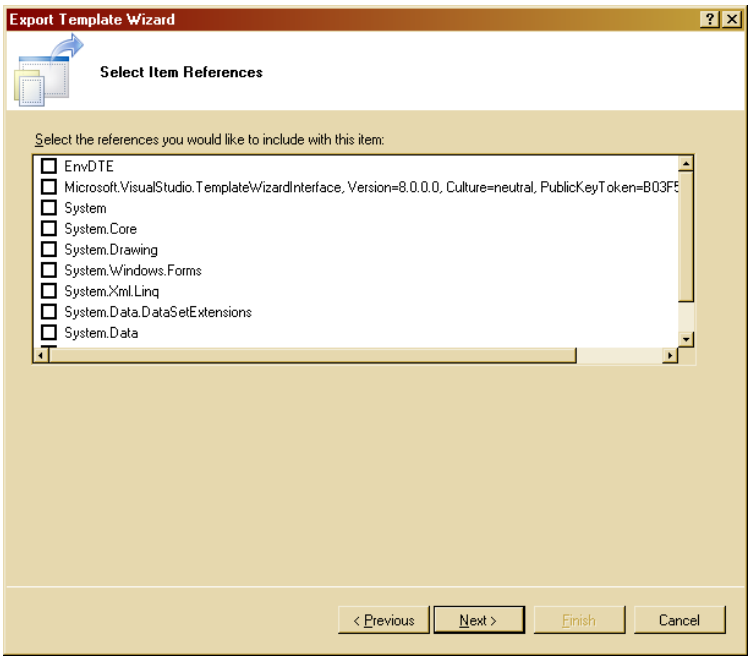
Obrázek 19: New Project (nová šablona a její popis)

Pokud si zvolíme „Item template“, tak uvidíme dialog (viz. Obrázek 20), kde si vybereme jednu třídu, kterou chceme mít jako „Item template“. Vždy jde vybrat pouze jedna třída resp. soubor, pokud se nejedná o fromulářovou třídu, která může být a většinou bývá

ve více souborech jako parcial class. Po stisknutí tlačítka Next se dostáváme na dialog (viz. Obrázek 21), kde je možnost výběru potřebných referencí pro náš zdrojový kód. Pokud nepotřebujeme žádnou referenci, stiskneme opět tlačítko Next a nabídku přeskočíme. Nyní už vidíme stejný dialog (viz. Obrázek 18), jako jsme již měli v případě projektové šablony. To znamená, že umíme vygenerovat jednoduché šablony, při jejich vytváření se do C# kódu doplnili některé proměnné, pro technologii template, o nichž jsme se zmínili již v předchozích kapitolách, aby byla možná jejich drobná variabilita v podobě přejmenování namespace podle názvu projektu u „Project template“, nebo změnu jména třídy u „Item template“. Takto vytvořené šablony by nám umožňovali jen omezené použití, a proto existuje možnost obohatit vygenerovaný projekt o vlastního průvodce.



Obrázek 20: Export template (volba části pro „Item template“)



Obrázek 21: Export template (výběr referencí)

3.5 Vlastní průvodce

Slouží pro uživatelské upravení konkrétní šablony. Může se například jednat o jednoduché přejmenování proměnných, metod nebo tříd, o vložení nebo nevložení částí kódu, případně i celých tříd. Průvodce nepracuje s konkrétní šablonou, ale s jejími proměnnými, takže je teoreticky možné jednoho průvodce použít pro více šablon za předpokladu, že budou použity všechny proměnné, které průvodce potřebuje.

3.5.1 Vytvoření vlastního průvodce

Vytvoření průvodce se skládá z několika kroků, které si pokusíme v následujícím textu jen stručně nastínit.

V prvním kroku bychom měli mít připravený projekt, pro který chceme průvodce připravit a měli bychom vědět, co chceme průvodcem dělat. Umístíme na konkrétní místa proměnné a vytvoříme template soubor podle návodu výše.

V druhém kroku si vytvoříme projekt pro samotného průvodce. Toto by měl být projekt typu Class Library. Vytvoříme formulář s prvky, kterými chceme řídit chování průvodce.

Ve třídě, která vznikla při vytváření projektu, je nutné implementovat rozhraní `Microsoft.VisualStudio.TemplateWizard.IWizard`.

Ve třetím kroku v metodě `RunStarted` vytvoříme volání vytvořeného formuláře a následné zpracování zadaných dat.

Ve čtvrtém kroku přidáme Strong Name Key (jedinečné jméno) proto, aby bylo možné přidat naši knihovnu do AssemblyCache. Přidání se provádí v properties projektu v záložce „Signing“.

V pátém kroku již vytvoříme build naší knihovny a přesuneme ji do složky „C:\WINDOWS\assembly\“ .

V šestém kroku přidáme do vygenerované šablony element `WizardExtension`, který zapříčiní, že se, při pokusu vytvořit naší šablonu, zavolá průvodce z assemblyCache, pokud je přítomen, pokud ne, tak se generování šablony zastaví chybovou hláškou. Element `WizardExtension` by mohl vypadat například takto:

```
  <WizardExtension>
    <Assembly> MyModuleSCWizardLib ,Version=1.0.0.0, Culture=Neutral,
PublicKeyToken=539e72478b5f6a7f
    </Assembly>
<FullClassName>MyModuleSCWizardLib.MyModuleSCWizard</FullClassName>
  </WizardExtension>
```

3.6 Shrnutí

Projekt založený pod VS 2008 nefunguje pod VS 2005, důvodem je odlišný název proměnné v atributu Project v elementu Import v souboru *.csproj. V opačném případě není problém. Ve VS 2008 se proměnná jmenuje `MSBuildToolsPath` a ve VS 2005 `MSBuildBinPath`.

Některé elementy se chovají ve VS 2005 a VS 2008 různě, například element `EnableLocationBrowseButton` sloužící ke znepřístupnění tlačítka „Browse“ ve VS 2005 skutečně funguje na toto tlačítko, na rozdíl od VS 2008, kde se znepřístupní celý řádek „Location“. Tento na první pohled drobný problém vede k tomu, že takový projekt není možné v tomto VS 2008 vůbec založit.

Vytvářet multi projekt bez podpory průvodce je, jak se při vytváření šablon ukázalo, ve složitějších případech nemožné. Například pokud má být nově vytvářený projekt součástí většího celku, je prakticky nemožné toto provádět.

Vytváření a ladění průvodce je při složitější aplikační logice velmi náročné, protože nelze ladit (debugovat) jeho chování vzhledem k nutnosti mít knihovnu umístěnou v assemblyCache.

4 T4 šablony

Vytváříme-li při zakládání nového projektu objekty, které jsou popsané nějakou strukturou (jež je možno parsovat jako například XML, nebo strukturovaný textový soubor nebo použít databázové tabulky), tak proč si potom nevytvořit technologii, která nám toto vytvoří sama, a my potom jen pomocí parciálních tříd nedopíšeme zbylý aplikační kód? Cesta, jak toto provést, samozřejmě existuje, může jí být například použití T4 (Text Templating Transformation Toolkit).

4.1 Obecné informace

T4 šablony je technologie, která je vyvinutá společností Microsoft a používaná pro generování kódu například v Microsoft DSL Tools (o této technologii generování kódu se zmíníme v další kapitole). Můžeme je najít i v *Microsoft Patterns & Practices Guidance Package*, jako *Smart Client Software Faktory* nebo *Web Client Software Faktory*.

Pomocí T4 šablon lze generovat kód pro C#, Visual Basic, XML i HTML a celou řadu dalších jazyků.

4.2 Použití

Abychom mohli T4 šablony začít používat, je nutné do VS 2005 doinstalovat SDK, ve VS 2008 a vyšších verzích, je již podpora pro T4 šablony implicitně zavedena. Ovšem ani v jednom případě Visual Studio neobsahuje InteliSense, takže máme vlastně k dispozici systémově náročný textový editor. Toto je možné změnit doinstalováním některého z nástrojů podporující zvýraznění syntaxe a i již zmiňovaný InteliSense. Tímto nástrojem může být například *Visual T4 Editor*[1], k dispozici je jak free verze, tak placená professional. Je možné také použít nástroj *T4 Editor plus UML-Style modeling tools*[2], který obsahuje i UML modelovací nástroj, který rovněž obsahuje free i professional verzi.

T4 šablony je možné použít v jakémkoliv projektu. Generování kódu z šablony zajišťuje VS a při kompilaci je použit pouze vygenerovaný kód, nikoli šablona.

---

[1] http://www.visualt4.com/

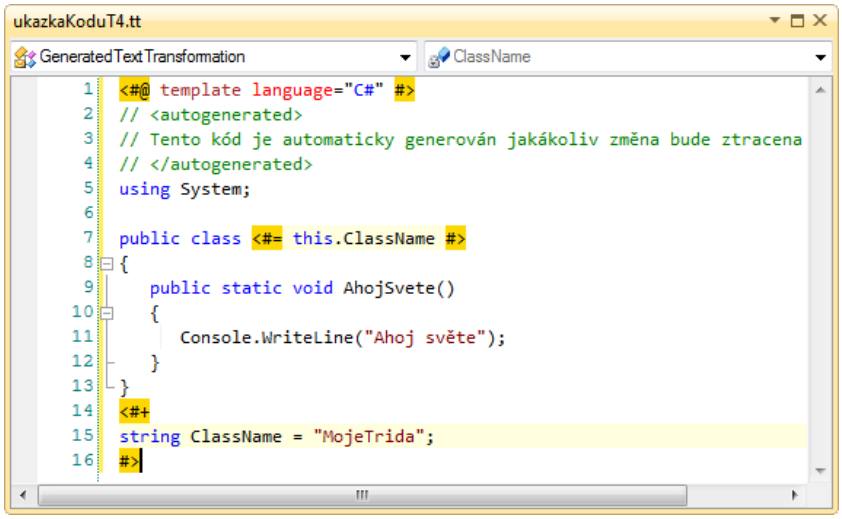
[2] http://t4-editor.tangible-engineering.com/T4-Editor-Visual-T4-Editing.html

4.3 Popis šablony

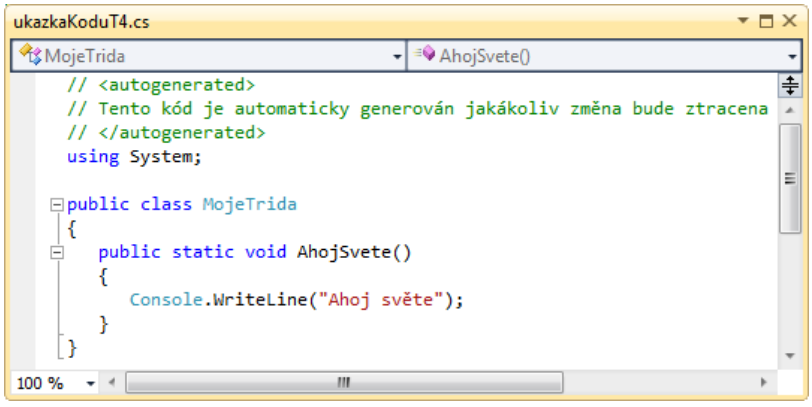
Šablona tohoto nástroje neobsahuje tolik tagů a možností nastavení jako předchozí, za to je ale více komplexní. Pomocí programu napsaném v C# nebo ve VB jsme schopni generovat jakýkoliv jiný programový kód a nemusí jít jen o .NET jazyky, ale může to být třeba SQL nebo i PHP, Python a mnoho dalších. Případně je možné generovat i neprogramové soubory jako HTML, XML nebo i obyčejný TXT.

Pro používání T4 šablon potřebujeme znát pouze několik nových klíčových slov (XML elementů) a značek a pak buď umět programovat v C#, nebo VB.

Princip použití je jednoduchý, napíše se šablona (viz. Obrázek 22) a po jejím uložení se vygeneruje výstup (viz. Obrázek 23).



Obrázek 22: Ukázka T4 šablony



Obrázek 23: Výsledek po vygenerování T4 šablonou

4.3.1 Klíčové slovo <#@ template #>

Syntaxe použití:

```
<#@ template [language="VB"] [hostspecific="true"] [debug="true"]
[inherits="templateBaseClass"] [culture="code"] #>
```

`Template` element obsahuje několik atributů, ale zmíníme se jen o těch nejdůležitějších.

Language

Specifikuje jazyk a jeho verzi, ve kterém se bude psát obslužný kód šablony. Na výběr je ze dvou možností C# a VB.

Příklad použití:

```
<#@ template language="C#v3.5" #>
```

Debug

Tento parametr nám dává možnost ladit šablonu, pokud nastavíme hodnotu parametru na True. Ladění ale znamená, že můžeme vidět vygenerovaný zdrojový kód, samozřejmě v jazyce, jaký jsme nastavili v předchozím parametru. Spolu s kódem se vygeneruje dalších 6 souborů. Všechny soubory mají stejné jméno, ale rozdílnou koncovku. Jméno se generuje po každém přeložení znovu a skládá se z osmi znaků. Složku se soubory nalezneme na této adrese: „TEMP folder (%USERPROFILE%\Local Settings\Temp)“. Soubory mohou vypadat takto:

```
csqg2m5s.0.cs, csqg2m5s.cmdline, csqg2m5s.dll, csqg2m5s.err,
csqg2m5s.out, csqg2m5s.pdb, csqg2m5s.tmp
```

Inherits

Definuje třídu, která náš soubor bude používat a bude řídit generování šablony. Třída, která bude používat šablonu, musí implementovat rozhraní `TextTransformation`.

4.3.2 Klíčové slovo <#@ output #>

Syntaxe použití:

```
<#@ output extension=".fileNameExtension" [encoding="encoding"] #>
```

Pomocí `output` elementu se nastavují parametry pro generování výstupního souboru. Všechny atributy jsou volitelné.

Extension

Atribut Extension slouží pro definování koncovky. Jakou má soubor koncovku, ještě nezajistí odpovídající obsah, ten musí zajistit autor šablony.

Příklad použití:

```
<#@ output extension=".cs" #>
```

Encoding

Atribut Encoding se používá na určení, jaké má být použito kódování pro generování souboru. Pokud se neurčí jinak, použije se kódování, které je nastavené v operačním systému.

Příklad použití:

```
<#@ output encoding="utf-8"#>
```

4.3.3 Klíčové slovo <#@ assembly #>

Syntaxe použití:

```
<#@ assembly name="[assembly strong name|assembly file name]" #>
```

Assembly element slouží pro načtení assembly, aby jej bylo možné použít. Obdobně jako je tomu ve VS při přidávání referencí do projektu. Jména je možno zadávat dvěma způsoby.

První způsob je použití silného jména (Strong name) z GAC, jako například System.Xml. Můžeme použít i dlouhý způsob zápisu ve formátu silného jména i se specifikací verze.

```
<#@ assembly name="System.Xml" #>
<#@ assembly name="System.Xml, Version = 4.0.0.0" #>
```

Druhý způsob použití je zadat assembly absolutní cestou. Toto použití je však nevhodné při týmovém vývoji, protože každý programátor může mít odlišnou adresářovou strukturu.

```
<#@ assembly name="C:\Projects\bin\Debug\ClassLibrary.dll" #>
```

4.3.4 Klíčové slovo <#@ import #>

Syntaxe použití:

```
<#@ import namespace="namespace" #>
```

Pomocí elementu import, lze odkazovat na objekty bez vypisování celého jména. Jedná se o ekvivalent z C#, a to konkrétně o using. I tento element lze použít několikrát.

Příklad použití:

```
<#@ import namespace="System.Diagnostics" #>
```

4.3.5 Klíčové slovo <#@ include #>

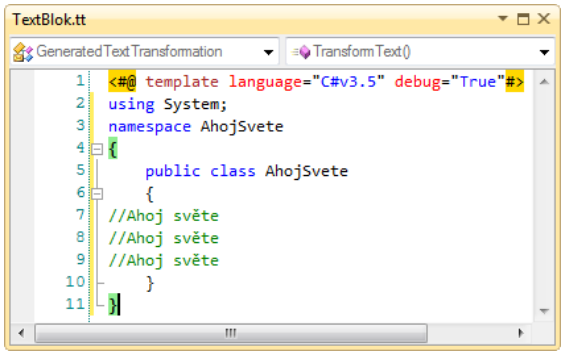
Element include slouží k vložení obsahu jiného souboru, jako by byl součástí souboru, ve kterém je použit tento příkaz. Element je možné použít opakovaně, a tak lze přiložit i několik souborů. Cestu je možno zadat jak absolutně, tak relativně.

Příklad použití:

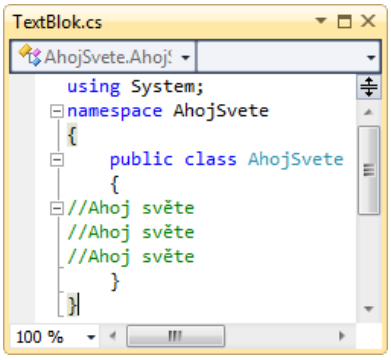
```
<#@ include file="c:\test.txt" #>
<#@ include file="Folder\Included.tt" #>
```

4.3.6 Textový blok

Textový blok je nějaký neprogramový text šablony. Obsah bloku se vypíše rovnou na výstup. Využívá se zejména při definování základní kostry šablony, jako je například definice using nebo namespace.



Obrázek 24: Ukázka textového bloku

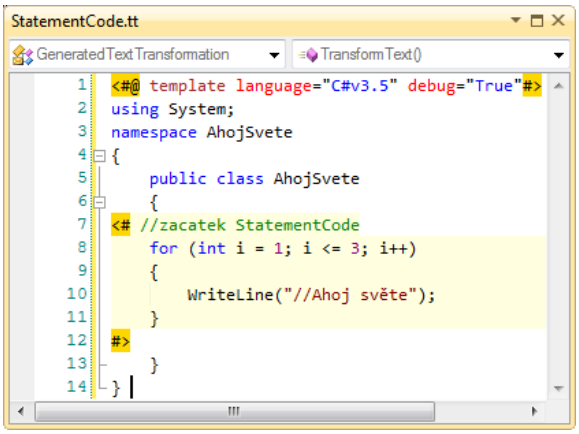


Obrázek 25: Výstup z T4 šablony, kde je jen textový blok

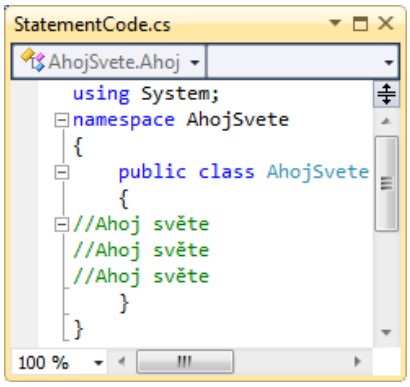
Obrázek 26: Zkompilovaná T4 šablona, kde je jen textový blok

4.3.7 Statement blok <# StatementCode #>

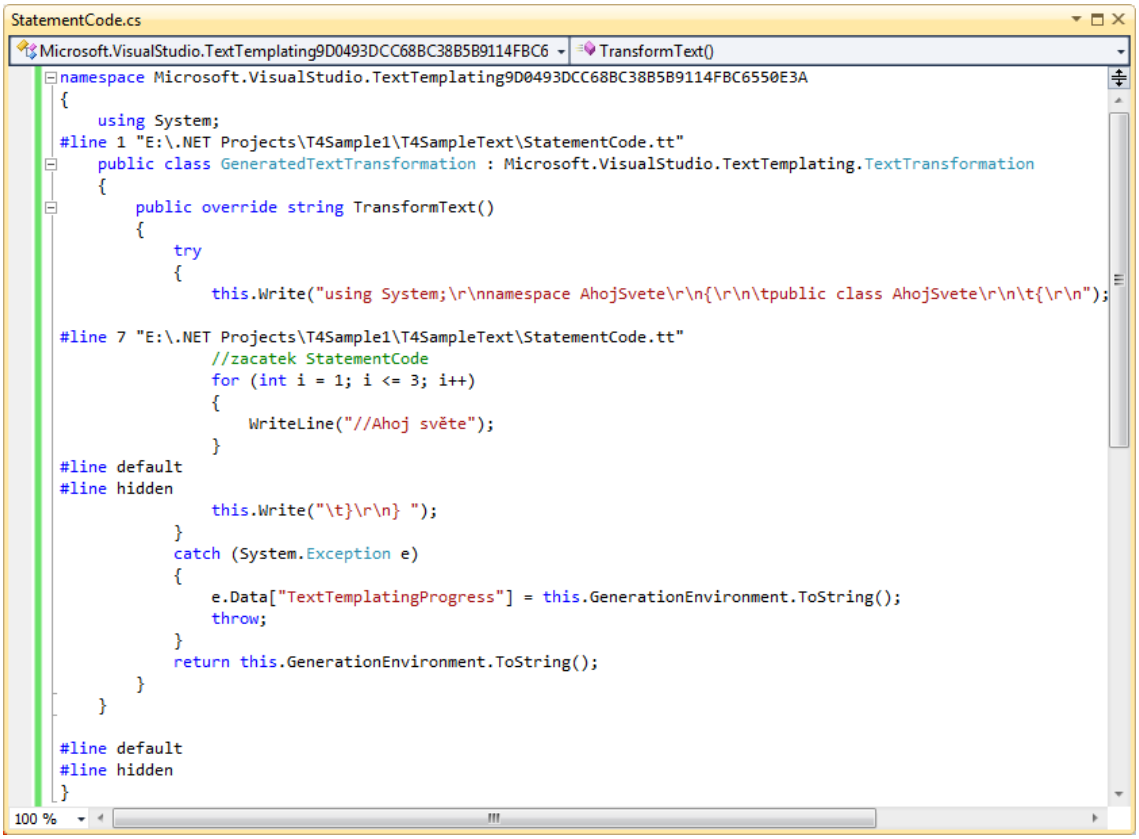
Do statement bloku se píše programový text šablony, který umožňuje psát základní syntaktické konstrukce jakou jsou cykly a větvení.



Obrázek 27: Ukázka Statement bloku



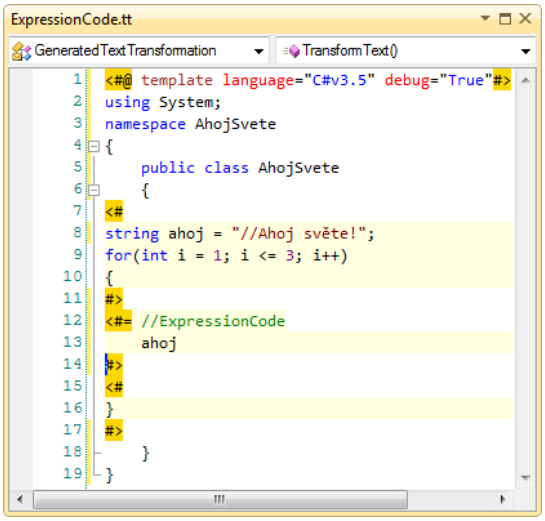
Obrázek 28: Výstup z T4 šablony, kde je Statement blok



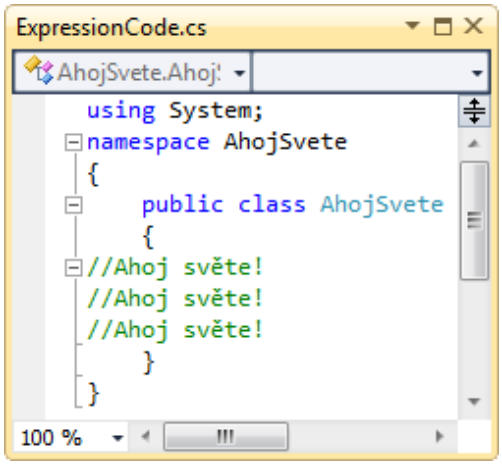
Obrázek 29: Zkompilovaná T4 šablona, kde je Statement blok

4.3.8 Expression blok <#= ExpressionCode #>

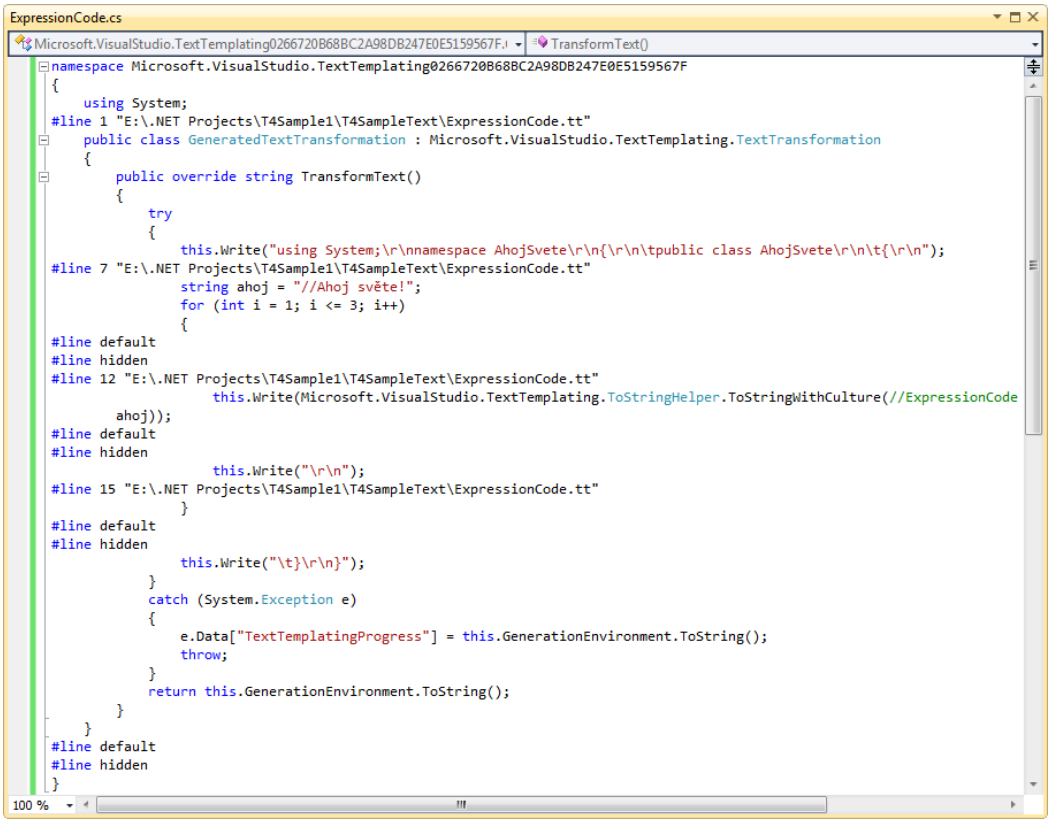
Tento blok slouží k výpisu hodnoty proměnné, čímž vlastně může vznikat dynamický kód po přegenerování šablonou.



Obrázek 30: Ukázka Expression bloku



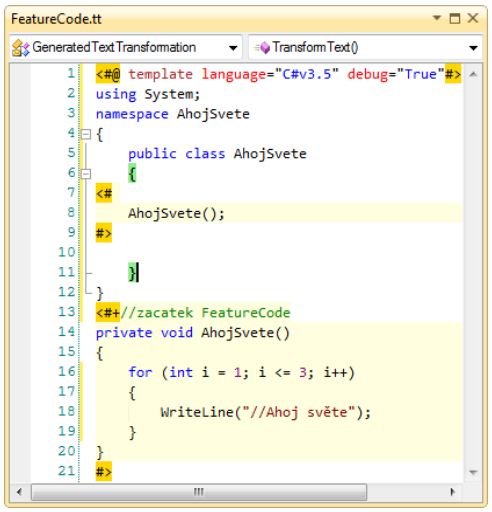
Obrázek 31: Výstup z T4 šablony, kde je Expression blok



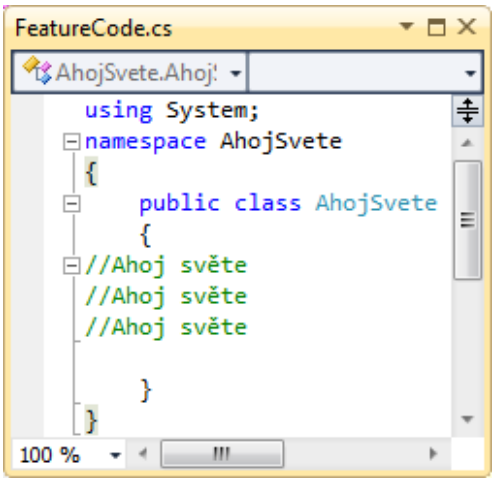
Obrázek 32: Zkompilovaná T4 šablona, kde je Expression blok

4.3.9 Feature blok <#+ FeatureCode #>

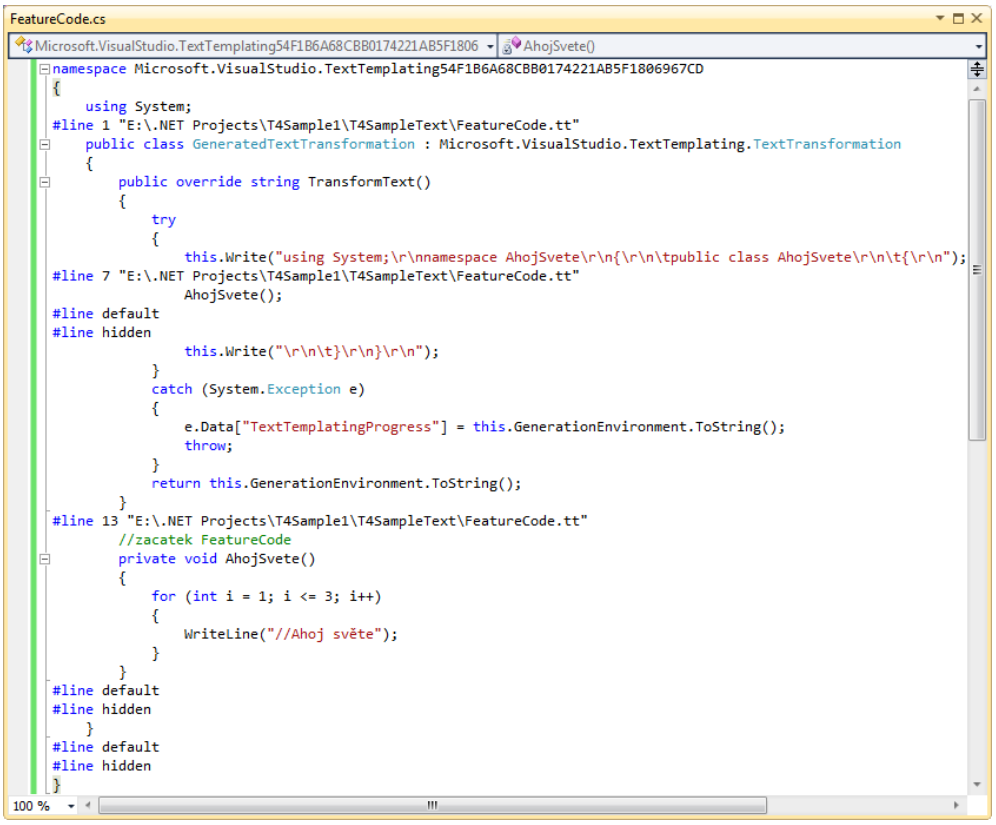
Tento blok se využívá pro přidání pomocné funkcionality při vytváření šablony. Pomocná metoda se použije pro opakované volání v hlavním kódu.



Obrázek 33: Ukázka Feature bloku



Obrázek 34: Výstup z T4 šablony, kde je Feature blok



Obrázek 35: : Zkompilovaná T4 šablona, kde je Feature blok

4.4 Novinky od Visual Studia 2010

Jednou z důležitých novinek ve VS 2010 je skutečnost, že je možné kód šablony použít i za běhu programu a ne jen při kompilaci. To je možné díky novým službám v API a novému enginu API, který tuto funkci podporuje. T4 šablony lze nyní použít i ve „Web Site Project“. Kompletní přehled změn sepsal na MSDN blogu Gareth Jones[3].

4.5 Komunitní nástroje

T4 Editor plus UML-Style modeling tools
http://t4-editor.tangible-engineering.com/T4-Editor-Visual-T4-Editing.html

T4 Toolbox
http://www.codeplex.com/t4toolbox

[3] http://blogs.msdn.com/b/garethj/archive/2010/04/15/what-s-new-in-t4-in-visual-studio-2010.aspx

5 Domain-Specific Language

Jak již z názvu Domain-Specific Language (doménově specifický jazyk) vyplývá, jedná se o jazyk vytvořený pro ucelenou oblast chování nějakých objektů. Kde reálný objekt je reprezentován objektem grafickým nebo textovým, a to i s jeho vlastnostmi a vazbami na ostatní objekty v dané oblasti.

DSL jazyky, jak již bylo nastíněno, mohou být buď grafické, nebo textové. Mezi textové DSL můžeme zařadit například regulární výrazy a SQL. Mezi grafické DSL patří LINQ to SQL a Entity Framework.

My se zde budeme věnovat hlavně grafickému DSL, a to ve formě Domain-Specific Language Tools.

Domain-Specific Language Tools(DSL Tools) jsou součástí Visual Studia SDK, a to již od verze VS 2005. Jedná se v podstatě o DSL jazyk, pomocí kterého si můžeme sami definovat vlastní DSL jazyk.

Výhody DSL jsou v tom, že jsou navrženy pro konkrétní účel a jsou pro něho i optimalizovány, aby co nejjednodušší a nejefektivnější formou řešily daný problém oproti obecnějším jazykům, jako je C# nebo VB. Na druhou stranu jeho nevýhodou je to, že pokud chceme v DSL vytvořit něco, pro co nebylo primárně určeno, pravděpodobně to nepůjde nebo půjde, ale velice obtížně a složitě.

Problematika vytváření DSL jazyků ve VS je natolik rozsáhlá, že se zde spíše seznámíme s některými možnostmi tvorby DSL.

5.1 Použití

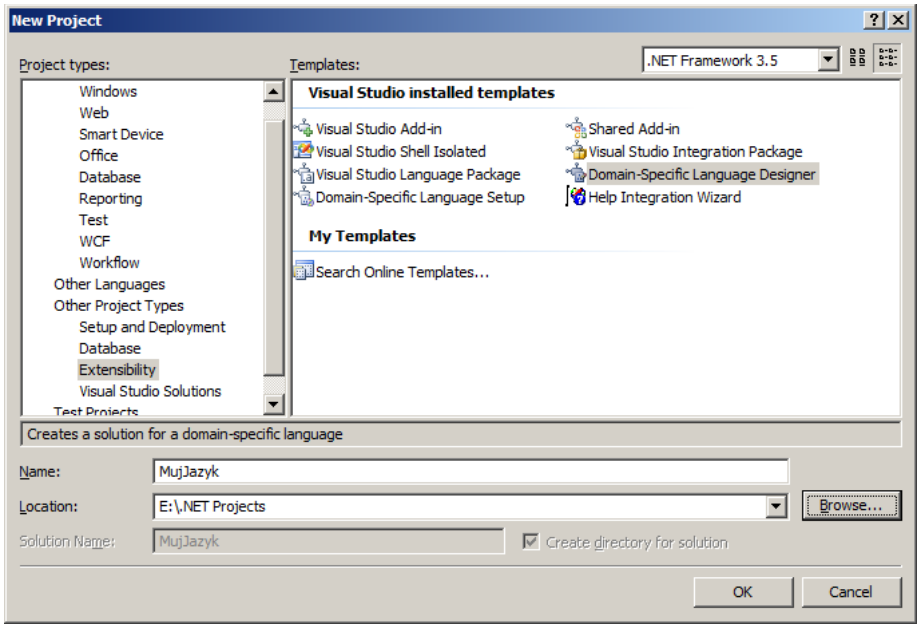
DSL má širokou škálu použití od generování kódu, který usnadní vývoj programátorům, až po vytváření prototypu aplikace, kterou lze prezentovat zákazníkovi. Pomocí DSL si můžeme připravovat modely potřebné pro naši aplikaci a následně z nich generovat SQL skript, programové objekty, ale třeba i dokumentaci.

Zjednodušeně lze postup tvorby a použití popsat takto: nejprve se vytvoří model modelovacího nástroje, poté je možno k němu vytvořit validace. V dalším kroku se začne vytvářet pomocí T4 šablon výsledný generátor kódu z nadefinovaného modelu. Nakonec stačí vytvořit instalační balíček pro VS a můžeme naše DSL distribuovat mezi vývojáře. Ovšem tato cesta není zdaleka tak jednoduchá, jak by z tohoto popisu mohla vypadat.

5.2 Založení DSL projektu

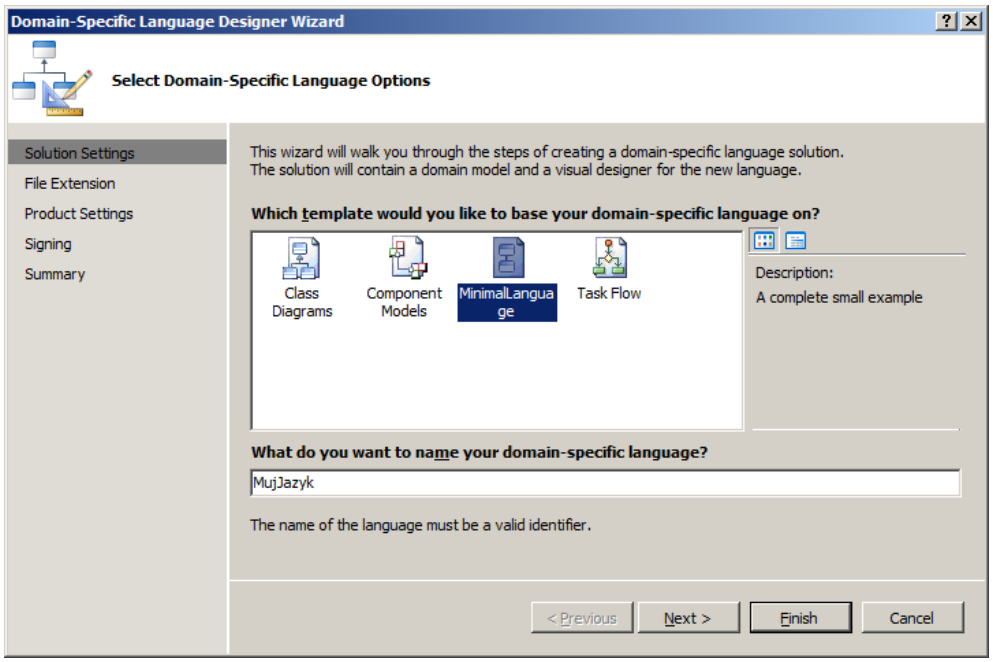
Abychom mohli založit náš první DSL projekt ve VS 2008, je potřeba mít nainstalované Visual Studio 2008 verze Standar Edition a vyšší se Service Pack 1 a Visual Studio 2008 SDK. Instalaci je nutno provést v tomto pořadí. Pokud není nainstalovaný Service Pack, tak není možné nainstalovat SDK. Pro založení projektu ve VS 2010 je potřeba nainstalovat Visual Studio 2010 verzi vyšší než Express, Visual Studio 2010 SDK a Microsoft Visual Studio 2010 Visualization & Modeling SDK.

Pokud máme vše nainstalováno, tak otevřeme VS a zvolíme „File > New > Project“ a měl by se zobrazit dialog „New Project“, na kterém vybereme položku „Other Project Types“ a následně „Extensibility“ (viz. Obrázek 36). Ještě vybereme v pravé části dialogu „Domain-Specific Language Desigener“ a vyplníme jméno našeho projektu.



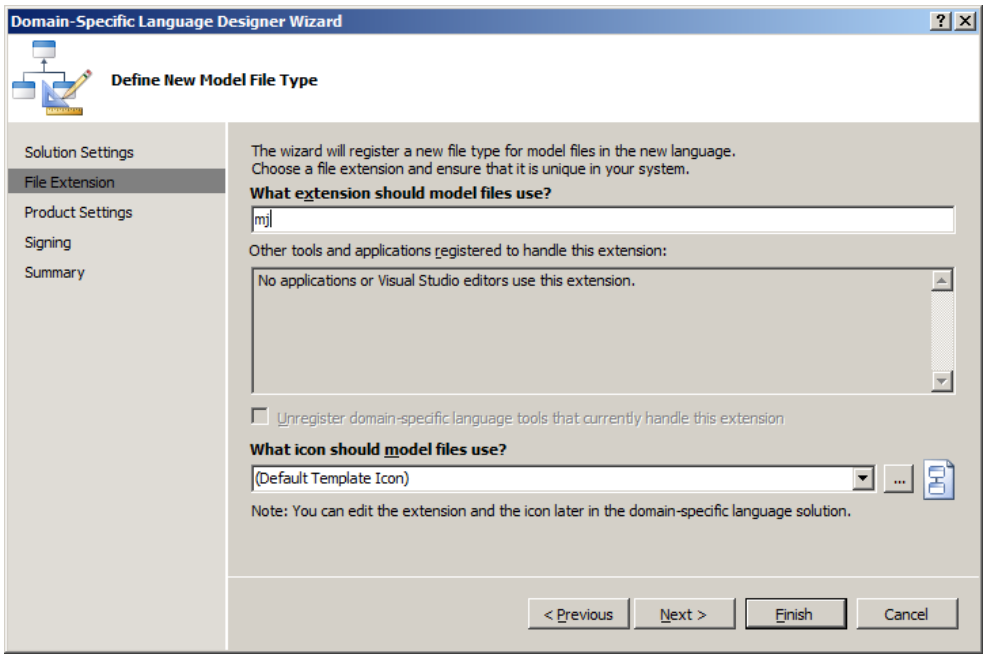
Obrázek 36: Dialog New Project s vybraným typem projektu

Nyní se spustí průvodce pro vygenerování našeho nového DSL, kde ve čtyřech krocích nastavíme parametry nového jazyka. Jako první si vybereme druh našeho jazyka. Zvolíme „MinimalLanguage“ a pojmenujeme si ho (viz. Obrázek 37). Ostatními možnostmi jako je „Class Diagrams“, „Component Models“ a „Task Flow“ se zde zabývat nebudeme, protože již obsahují připravené modely pro rychlejší vývoj v konkrétní oblasti. Tyto projekty je možno vytvořit i z „MinimalLanguage“.



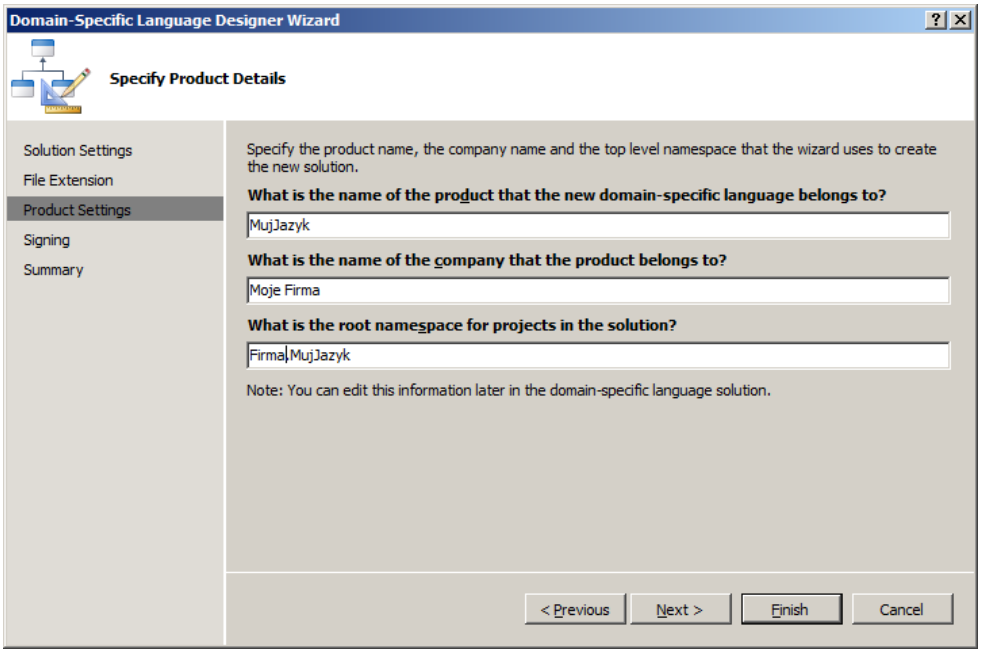
Obrázek 37: Průvodce vytvoření DSL. Vybrání typu DSL.

Jako další je potřeba zvolit příponu, kterou bude mít náš DSL jazyk, a jeho ikonu. Nastavení těchto parametrů bude vidět až při používání hotového DSL ve Visual Studiu. Toto nastavení a všechna následující jsou již pro všechny čtyři typy jazyka stejné.



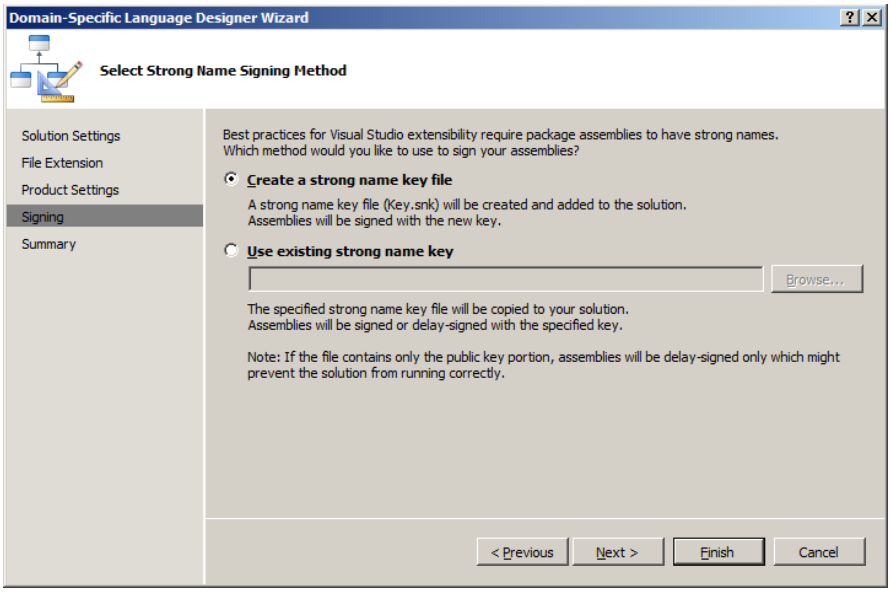
Obrázek 38: Průvodce vytvoření DSL. Definování modelu DSL.

V třetím dialogu se vyplňuje jméno nového DSL jazyka a v druhém řádku je požadován název firmy (viz. Obrázek 39). Jméno DSL a firmy se následně použije do Assembly Imformation založených projektů. Poslední je volba jmenného prostoru pro vygenerované projekty, ve kterých budeme vytvářet náš jazyk.



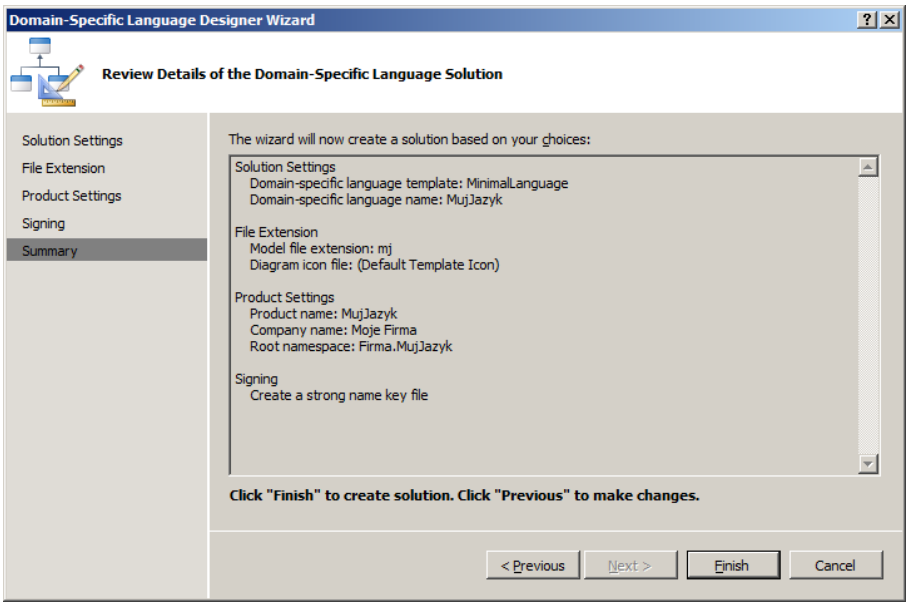
Obrázek 39: Průvodce vytvoření DSL. Specifikace konečného DSL.

Jako poslední se nastavuje silné jméno pro použití v hotových knihovnách (viz. Obrázek 40). Jsou zde dvě možnosti, buď ho necháme vytvořit automaticky při generování projektu, nebo přiřadíme již existující a při generování bude zkopírován do projektu.



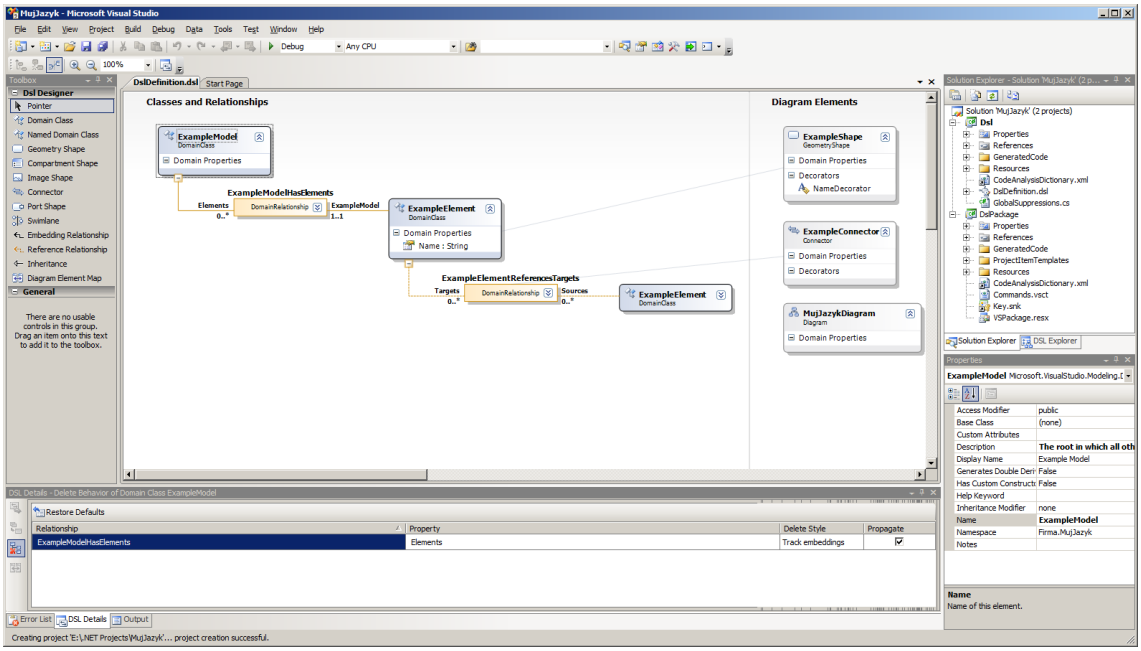
Obrázek 40: Průvodce vytvoření DSL. Volba vytvoření silného jména.

Závěrečné shrnutí našeho nastavení nám umožňuje zkontrolovat si, že jsme vše nastavili tak, jak jsme chtěli. Pokud tomu tak není, je možné se k předchozím krokům vrátit a opravit chybné nastavení. Je-li vše v pořádku, zbývá klinknout na tlačítko „Finish“ a projekt se vygeneruje. Pokud chceme ponechat výchozím nastavením, můžeme již na prvním dialogu (viz. Obrázek 37) po vybrání typu DSL stisknout tlačítko „Finish“.



Obrázek 41: Průvodce vytvoření DSL. Závěrečné shrnutí nastavení.

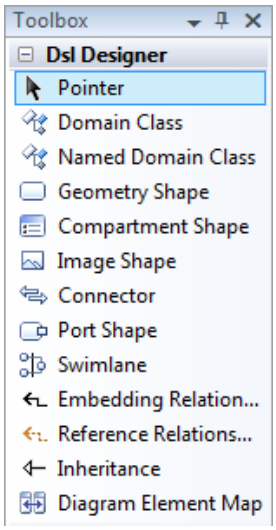
Po vygenerování projektu bychom měli vidět ve Visual Studiu předgenerovaný projekt (viz. Obrázek 42). Jednotlivé části tohoto dialogu budou popsány v kapitole 5.3.



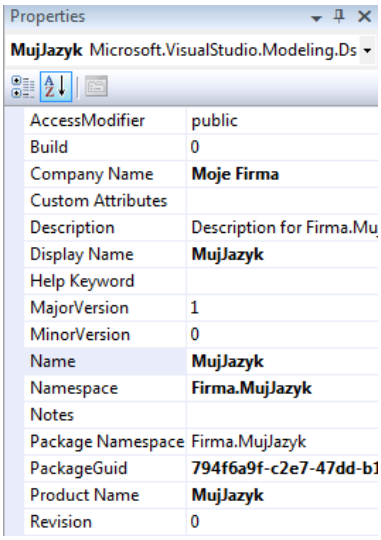
Obrázek 42: Visual Studio po vygenerování nového projektu typu DSL MinimalLanguage

5.3 *Popis jednotlivých částí DSL projektu*

Nyní si popíšeme, z jakých částí se skládá okno VS pro práci s DSL projekty. Některé části jsou podobné jako v jiných typech projektu, jako například „Toolbox“ (viz. Obrázek 43), „Properties“ (viz. Obrázek 44) nebo „Solution Explorer“ (viz. Obrázek 45).

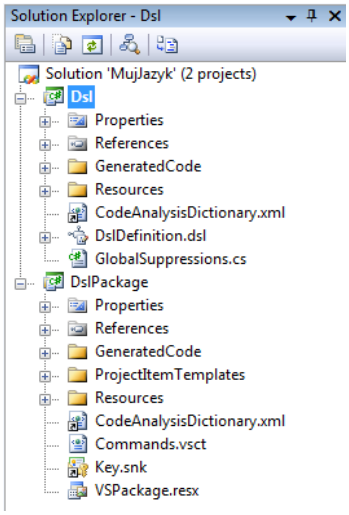


Obrázek 43: „Toolbox“ DSL projektu

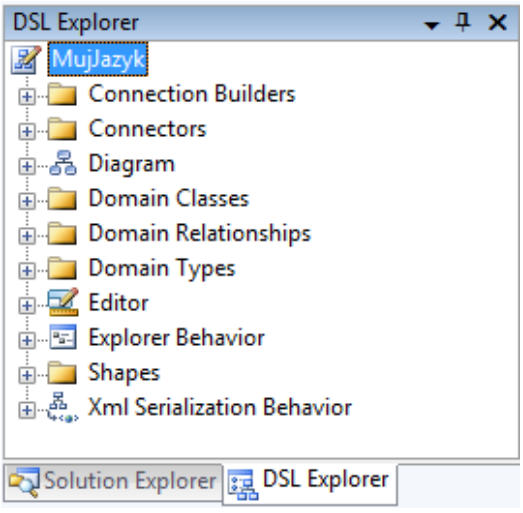


Obrázek 44: „Properties“ (vlastnosti) DSL projektu

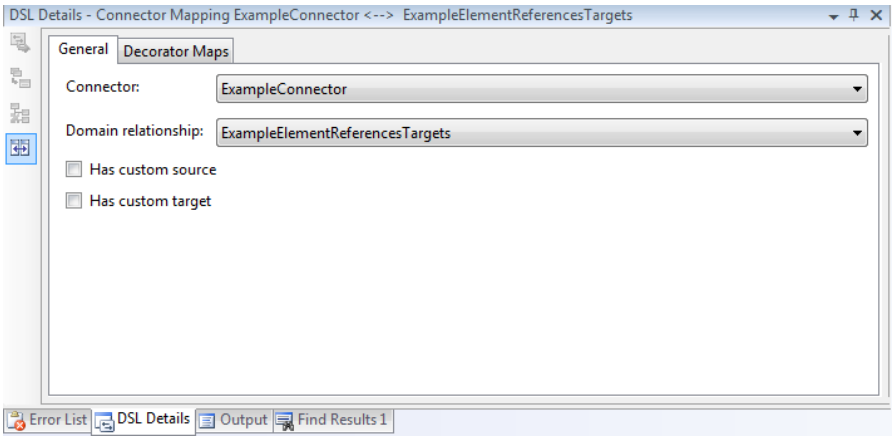
Jiné části VS jsou odlišné od všech ostatních projektů, mezi ně patří například „DSL Explorer“ (viz. Obrázek 46) nebo „DSL Details“ (viz. Obrázek 47).



Obrázek 45: „Solution Explorer“ DSL projektu

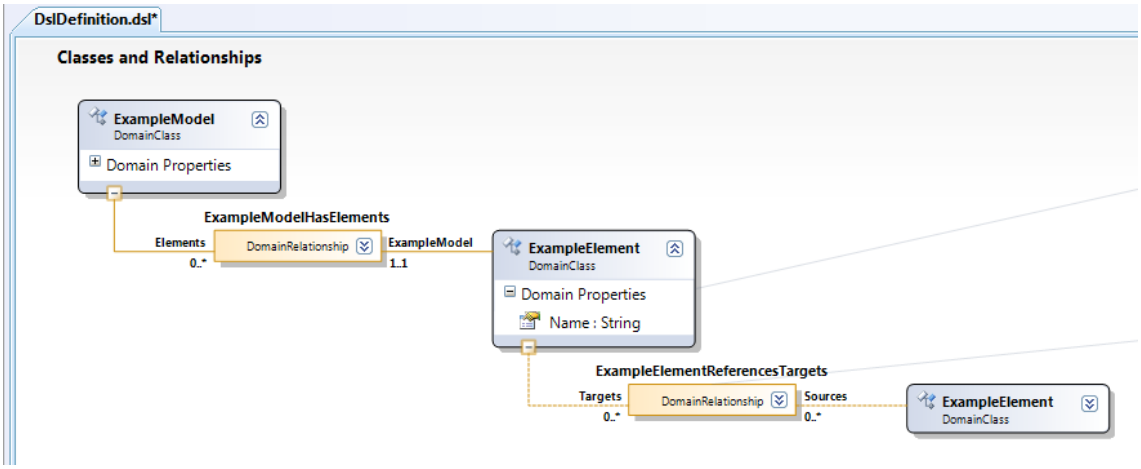


Obrázek 46: „DSL Explorer“

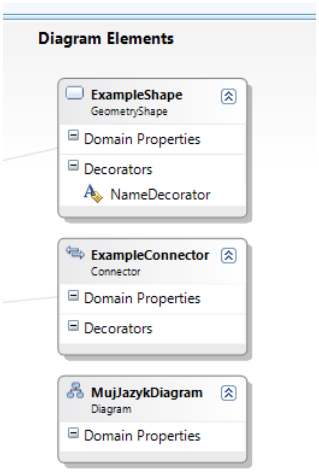


Obrázek 47: „DSL Details“

Ve zbytku okna VS je pracovní plocha pro modelování DSL jazyka, která je ještě rozdělena na dvě části. V levé části je oblast „Classes and Relationships“ (viz. Obrázek 48), kde se modelují objekty a vazby mezi nimi. Pravá část slouží pro nastavování vzhledu a zobrazování jednotlivých objektů a vazeb a je pojmenovaná „Diagram Elements“ (viz. Obrázek 49).

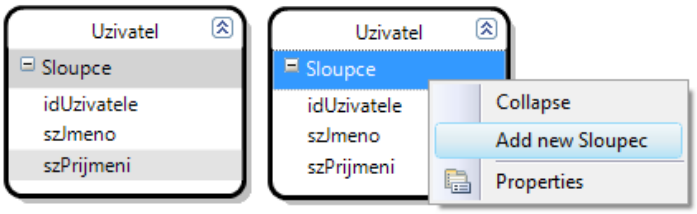


Obrázek 48: Oblast „Classes and Relationships“



Obrázek 49: Oblast „Diagram Elements“

Pro modelování DSL můžeme použít nástroje, které jsou vidět v panelu nástrojů (Toolbox). Máme tu k dispozici doménové třídy „Domain Class“, které reprezentují objekty v našem jazyce. Nebo je možné použít konkrétnější objekt „Named Domain Class“, což je „Domain Class“ s vlastností jméno (Name). Objekty je nutné mezi sebou propojovat, a to je zde možné pomocí tří vazeb. Lze vytvořit agregační spojení mezi objekty pomocí „Embedding Relationship“. Další možností je provést spojení prvků jednoduchou asociací, a to pomocí prvku „Reference Relationship“. Poslední vazbou pro propojení objektu je dědičnost, pro kterou je zde vazba Inheritance. Je důležité si uvědomit, že doménové třídy, kterými jsou i vazby, mají atributy. V metamodelu DSL mají vazby mezi objekty atributy, což představuje rozdíl oproti základnímu metamodelu pro UML. Všechny výše popsané prvky se používají v části „Classes and Relationships“. Následující čtyři prvky z panelu nástrojů se naopak vkládají do plochy „Diagram Elements“ a slouží pro grafické znázornění namodelovaných objektů. Těmito nástroji jsou „Geometry Shape“, „Image Shape“, „Compartment Shape“ a „Connector“. „Geometry Shape“ reprezentuje geometrický návrh v podobě obdélníků, obdélníků se zaoblenými rohy, elipsy atd. „Image Shape“ reprezentuje objekt jako obrázek. „Compartment Shape“ reprezentuje objekt jako obdélník nebo obdélník se zaoblenými hranami, ale navíc obsahuje rozbalovací část, na kterou lze namapovat jiný objekt, pomocí kterého je možno přidávat další vlastnosti (např. k databázové tabulce sloupce nebo ke třídě metody) (viz. Obrázek 50). Pomocí nástroje „Connector“ se nastavuje zobrazení vazeb mezi objekty. Propojení mezi logickou podobou objektu a vazeb s jejich grafickou podobou se provádí pomocí nástroje „Diagram Element Map“.



Obrázek 50: Příklad použití „Compartment Shape“

5.4 Shrnutí

Popisovat zde vytvoření modelu krok po kroku by bylo příliš zdlouhavé a ani tak by nebylo možné obsáhnout všechny možnosti tohoto nástroje. Srozumitelně tento postup popsal Jean-Marc Prieur v dokumentech *Visual Studio Visualization and Modeling SDK (DSL Tools)*, které jsou umístěny na internetové adrese (http://code.msdn.microsoft.com/DSLToolsLab).

Je několik možností, které lze s modelem dělat. K modelu můžeme vytvořit vlastní validační pravidla, která budou kontrolovat, jestli je model logicky správný. Model může načítat data z různých zdrojů a na jejich základě generovat vizuální reprezentaci, jak je tomu například u Entity Frameworku. Z modelu je možno generovat různé textové fragmenty pomocí T4 šablon, ať už to jsou například kusy kódu v jazyce C# či VB, nebo skriptovacího jazyka jako například T-SQL a nebo značkovacího jazyka jako je HTML.

DSL je velmi robustní nástroj pro pokročilou tvorbu automatického generování kódu a jeho vývoj je určen spíše zkušeným programátorům.

6 Guidance Package

Představme si, že jsme vytvořili návrh kostry modulární aplikace, kdy tuto kostru budou pro další vývoj potřebovat ostatní vývojáři. Pokud je s touto aplikací pouze seznámíme a necháme je kopírovat původní řešení, nedopadne to dobře. Bude se stávat, že někdo na něco zapomene nebo špatně upraví. Proto by bylo dobré vytvořit šablony, které podle vstupních parametrů vytvoří kostru a budou i schopné přidávat další artefakty do projektu a kódu. K většině těchto akcí by nám stačili nástroje zmíněné v předchozích kapitolách. My ale chceme jít s použitelností dál a integrovat naše nové nástroje pro vývoj přímo do kontextového menu ve VS. Přesně s tím nám může pomoci Guidance Package.

Pomocí Guidance Package můžeme vytvořit instalační balíček, který nám integruje do Visual Studia CodeSnippit, VS Template, T4 šablony a třeba i vlastní akce, které si napíšeme například v C#. Guidance Package používá vlastního průvodce, kterého lze uživatelsky definovat. Pokud potřebujeme například při generovaní nového souboru pomocí VS Template vstupní data, nemusíme si průvodce psát sami a lze si jen nadefinovat, co má dialog obsahovat, případně jak má validovat vstupy, a zda jsou povinné nebo ne.

Obsah Guidance Package se opět definuje pomocí jednoho XML souboru. Struktura tohoto souboru již není tak jednoduchá jako u předchozích technologií, a proto až si budeme popisovat přidávání jednotlivých prvků do Guidance Package, tak nebudeme popisovat jednotlivé elementy tohoto XML, ale pouze funkce jednotlivých bloků.

6.1 Použití

Pro vývoj Guidance Package (GP) je třeba mít nainstalovaný Guidance Automation Toolkit (GAT). Jakmile dokončíme práci na našem projektu a chceme ho distribuovat ostatním vývojářům, stačí z projektu vytvořit build, po kterém nám vznikne soubor s příponou *.vsix, pomocí kterého provedeme instalaci na ostatní počítače. Před jeho instalací je ale nutné nainstalovat Guidance Automation Extensions (GAX).

GP se nejčastěji používají pro podporu vývoje za pomocí softwarových továren, konkrétní použití je možno najít například ve „Smart Client Software Factory“[4] nebo „Web Client Software Factory“[5].

Základními stavebními prvky GP jsou `recipes` (recepty), `actions` (akce), VS šablony (viz. kapitola 3) a T4 šablony (viz. kapitola 4).

Recepty jsou automatizované opakující se činnosti, které by vývojář musel provádět ručně. Recepty obsahují akce a definují jejich pořadí, ve kterém se mají provádět. Kromě toho může recept obsahovat další recepty a šablony a používat hodnoty poskytované průzkumníky. Recept může být proveditelný jednou, v takovém případě zmizí z menu poté, co byl proveden, nebo proveditelný vícekrát, v takovém případě ho lze provádět opakovaně.

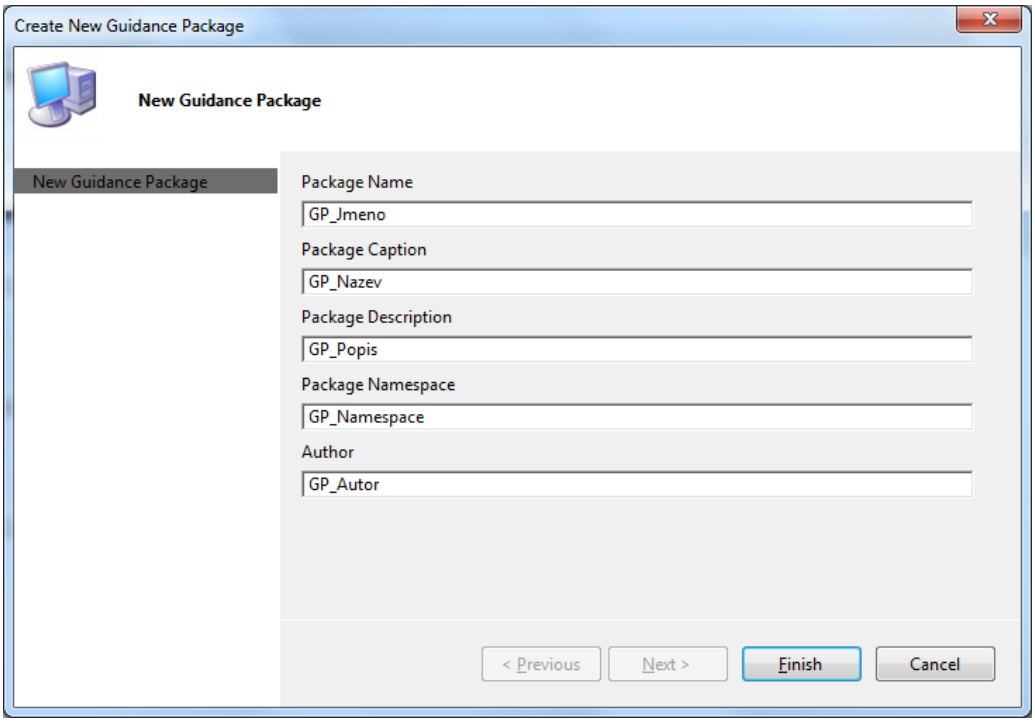
Akce jsou automatizované bloky používané recepty, které je vyvolávají. Akce přijímají vstupní data, která byla shromážděna receptem vyvolávajícím tuto akci, nebo jinou akcí, která proběhla před ní.

6.2 Postup založení nového Guidance Package

Než začneme vytvářet nový projekt, je nutno nejprve nainstalovat GAT. Založení nového projektu probíhá standardně spuštěním VS, vybráním nabídky „File > New > Project…“. V dialogu „New Project“ vybereme položku „Guidance Package Development“ a následně „Guidance Package“. Po pojmenování projektu, zvolení umístění projektu a stisknutím tlačítka „OK“ se zobrazí dialog „Create New Guidance Package“. V tomto dialogu se nastavuje „Package Name“, což je identifikační jméno projektu, „Package Caption“ je jméno projektu, které se bude zobrazovat. „Package Description“ je popis nově vytvářeného balíčku. „Package Namespace“ určuje, jaký jmenný prostor se bude používat při vytváření nových tříd použitých v Guidance Package, nikoli při používání Guidance Package. Kromě Namespace lze tyto údaje měnit i dodatečně.

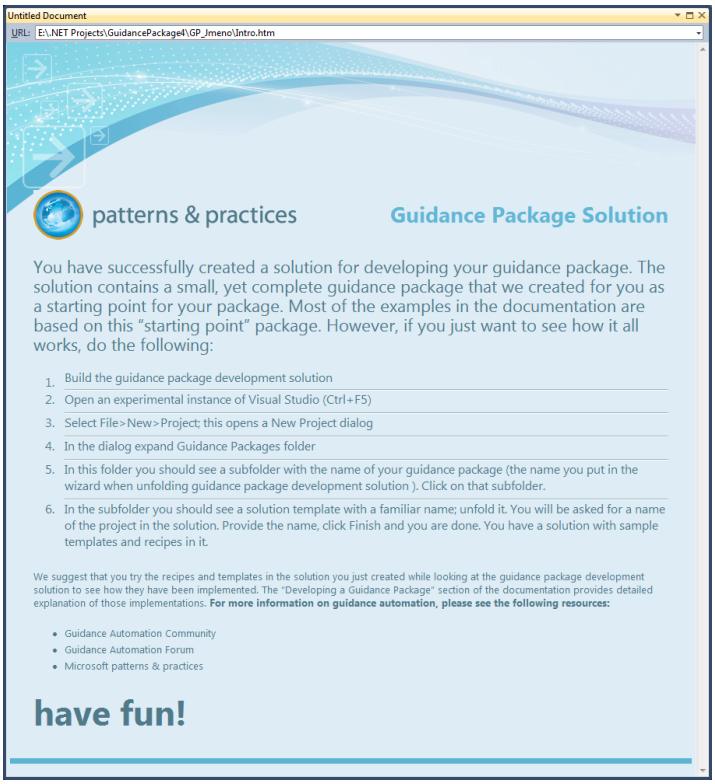
[4] http://msdn.microsoft.com/en-us/library/ff709810.aspx

[5] http://msdn.microsoft.com/en-us/library/ff699511.aspx

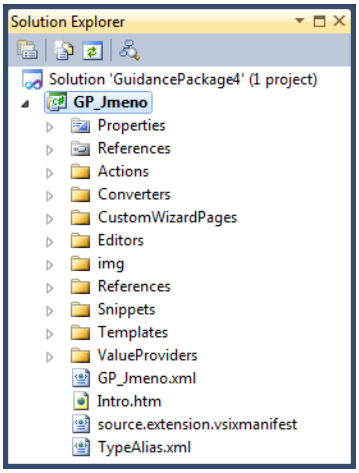


Obrázek 51: Dialog „Create Gudance Package“ při zakládání projektu

Po zadání všeho potřebného a stisknutí tlačítka „Finish“ se vygeneruje projekt, kde jsou ukázky některých operací, které je možno využít při vytváření vlastního řešení nebo je odmazat a začít s čistým projektem. Po vygenerování se zobrazí uvítací obrazovka (viz. Obrázek 52). Strukturu nového projektu obsahuje řadu složek a několik XML souborů (viz. Obrázek 53).



Obrázek 52: Uvítací obrazovka

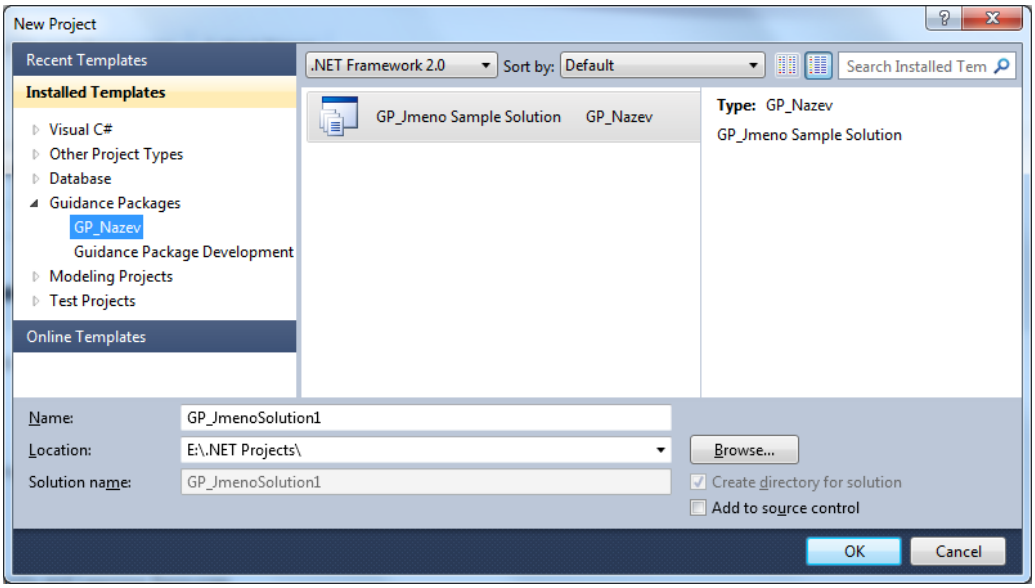


Obrázek 53: Struktura nového projektu

6.3 První spuštění nového Guidance Package

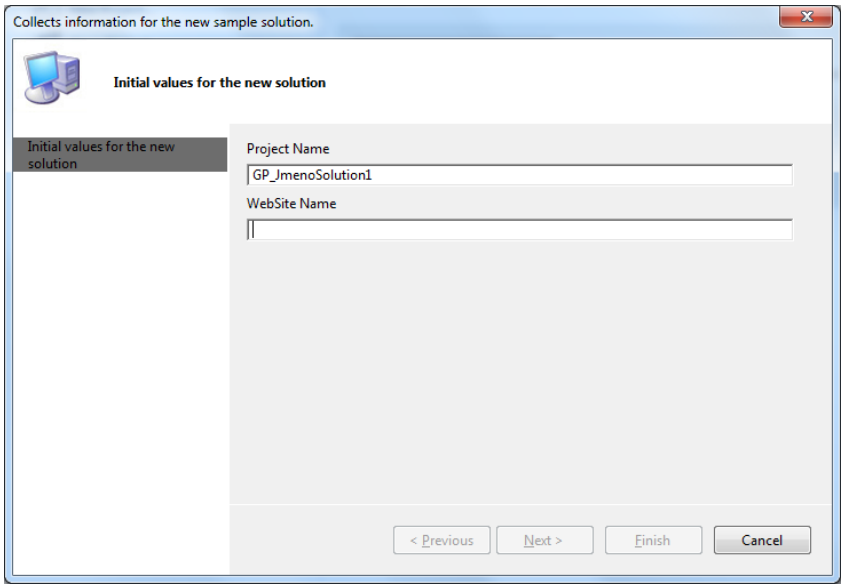
Takto vygenerovaný projekt je hned připraven k použití a lze ho spustit ve vývojové verzi VS. To znamená, že se spustí nová instance VS a v ní je v dialogovém okně

„New Project“ přidán náš vyvíjený balíček pod položkou „Guidance Packages“. Zde se zobrazují všechny balíčky, které máme nainstalovány. Výhoda této vývojové instance VS je v tom, že v ostrém VS v tuto chvíli není přidán vyvíjený balíček.



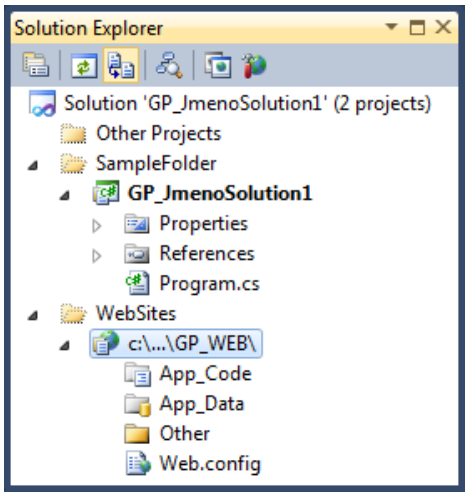
Obrázek 54: Dialogové okno „New Project“ s přidaným novým balíčkem

Přidaný projekt se zobrazuje v položce „Guidance Packages“ (viz. Obrázek 54), jehož založení bylo popsáno na předchozích řádcích. Nyní si ukážeme na několika obrázcích část toho, co nám připravili vývojáři GAT jako ukázky, v nově založeném projektu, pro vytváření našich Guidance Package.

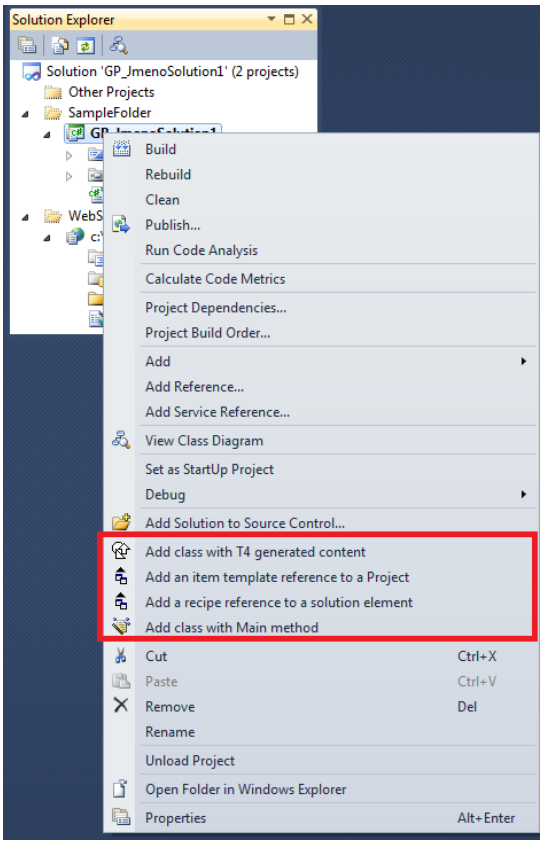


Obrázek 55: Dialog nově založeného Guidance Package

Na začátku nově zakládaného projektu z našeho GP je nastavovací dialog (viz. Obrázek 55). Vzhled tohoto dialogu je podobný jako při zakládaní GP balíčku, a je to proto, že i nový projekt Guidance Package je generován pomocí jiného GP. Základ tohoto dialogu je pevně daný a pouze se určuje obsah levé editační části, tzn., že můžeme určovat, jak se budou jmenovat popisy editačních polí, a to, jestli je to textové pole nebo rozbalovací nabídka, případně jiný editační prvek. Je vidět, že je možné nastavit, mají-li být prvky předvyplněné nebo povinné a tím ovlivnit chování dialogu, respektive tlačítek.

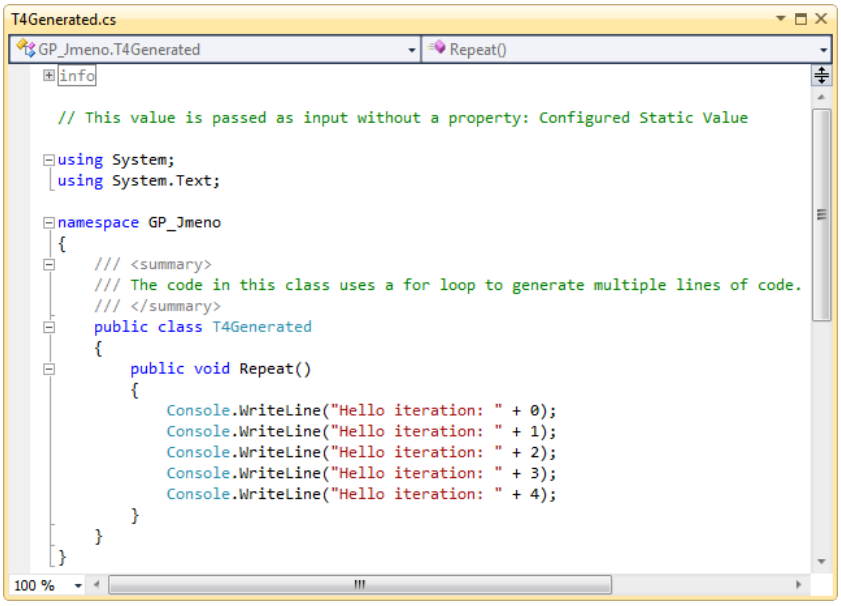


Obrázek 56: Založená „Solution“ z připraveného GP



Obrázek 57: Přidaný obsah kontextového menu

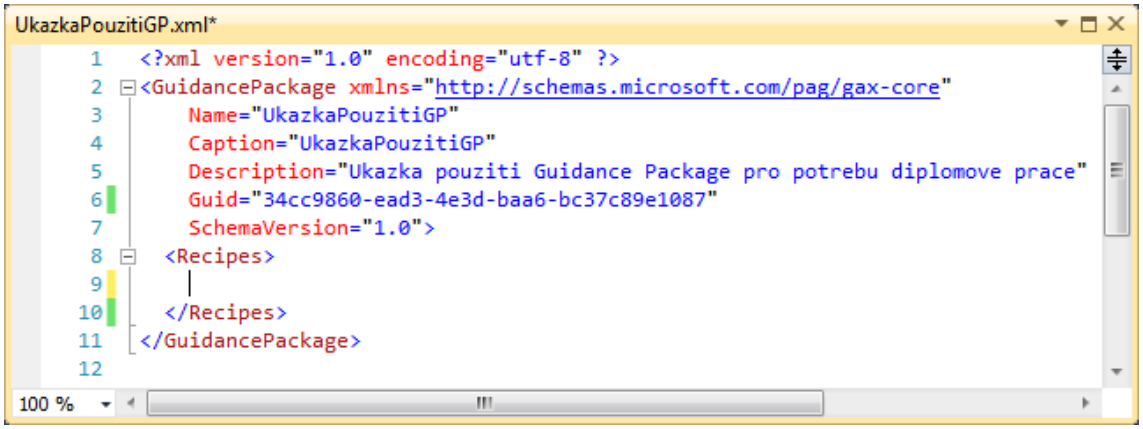
Výsledkem zakládaného projektu je vlastně celá „Solution“, která obsahuje tři složky a dva projekty (viz. Obrázek 56). Projekty jsou pojmenovány tak, jak jsme zvolili v nastavovacím dialogu (viz. Obrázek 55). GP rozšířil kontextové menu vyvolatelné nad jednotlivými projekty (viz. Obrázek 57). Jiná rozšíření by byla vidět v kontextovém menu pro „WebSite“ projekt, anebo v položce „Add“ v menu nově založených projektů. Například první ze zvýrazněných položek „Add class with T4 generated content“ vyvolá standardní dialog GP s textovými poli „Class Name“, „Namespace“ a „Iterations“ a po potvrzení zadaných hodnot v dialogu se založí třída (viz. Obrázek 58).



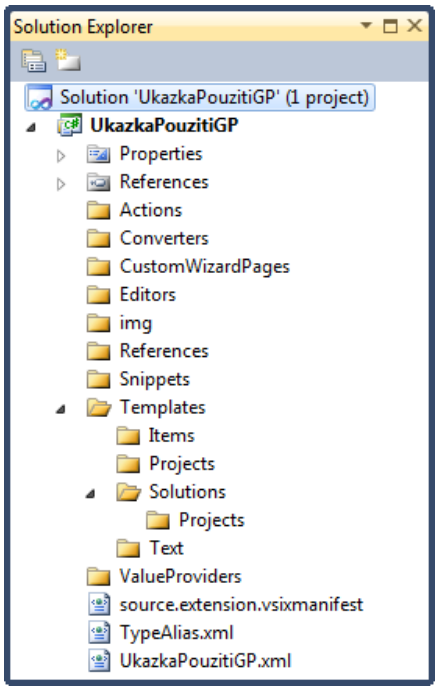
Obrázek 58: Třída vygenerovaná pomocí položky v kontextovém menu

6.4 Vytvoření vlastního balíčku

Než si začneme popisovat vytvoření vlastního balíčku s jednoduchým projektem, odmažeme si všechny ukázkové soubory (viz. Obrázek 60) a předpis celého Guidance Package (viz. Obrázek 59).

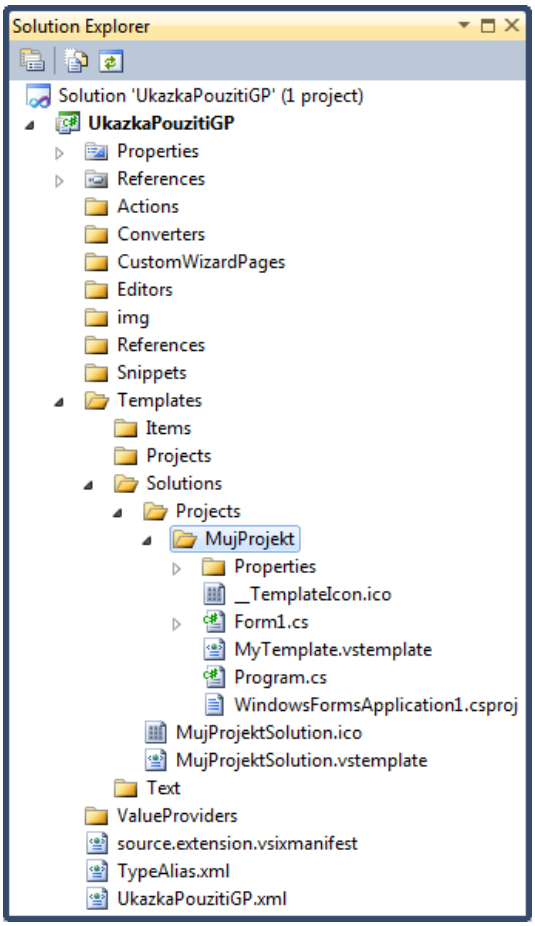


Obrázek 59: Promazaný konfigurační soubor Guidance Package

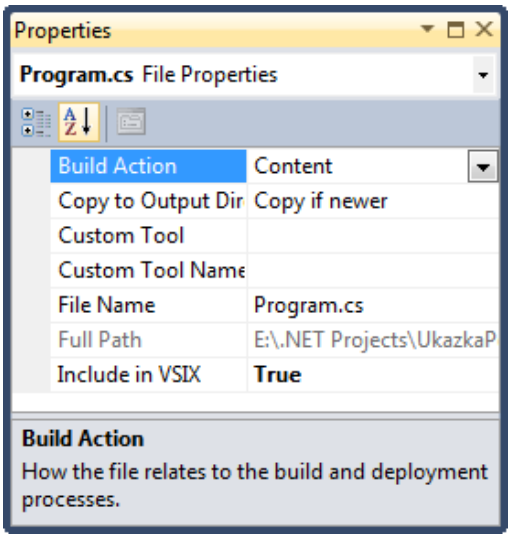


Obrázek 60: Promazaný projekt Guidance Package

Jakmile toto máme, je třeba si buď vyrobit nějaký projekt a z něho VS šablonu, nebo mít již nějakou VS šablonu hotovou. Vytvoření šablony si zde nebudeme popisovat, protože to bylo již součástí kapitoly 3. Zde použijeme jednoduchou šablonu, vytvořenou z projektu „WindowsFormsApllication“, kde jsme si do formuláře přidali tlačítko. Při vytváření tohoto nového projektu budeme chtít upravovat jeho jméno, jméno formuláře a text tlačítka. Vložíme naši šablonu do složky „Templates/Solutions/Projects“ (viz. Obrázek 61). Nyní je třeba vložit soubor *.vstemplate, který bude reprezentovat „Solution“, která se vytvoří při zakládání nového projektu a samozřejmě také ikonu, na kterou je potřeba do vstemplate souboru vytvořit odkaz. U všech souborů šablony je nutné nastavit vlastnosti tak, aby se při sestavování balíčku nepřekládaly, ale kopírovaly se do něho (viz. Obrázek 62). To znamená nastavit vlastnosti „Build Action“ (na „Content“), „Copy to Output Directory“ (na „Copy if newer“) a „Include in VSIX“ (na „True“). Nyní upravíme soubor „MujProjektSolution.vstemplate“, který představuje naši „Solution“ (viz. Obrázek 63). Ještě bude třeba upravit soubor vstempate se šablonou projektu, kde je třeba před ukončovací tag `</VSTemplate>` zkopírovat tag `<WizardExtension>` ze „Solution“ šablony s celým jeho obsahem, aby šablona věděla, odkud má přebírat data pro nahrazování proměnných.



Obrázek 61: Vložení šablony projektu do GP projektu



Obrázek 62: Ukázka nastavení vlastností souboru

Nyní jako poslední krok je třeba připravit konfigurační soubor GP (viz. Obrázek 64). V kódu je vidět definovaný recept se jménem `CreateSolution`, dále stojí za zmínku oblast `Arguments`, kde v jednotlivých argumentech jsou definovány a pojmenovány editační prvky dialogu. Tyto prvky mohou obsahovat různé validace a kontroly. V další části receptu je vidět definice vlastního průvodce, která je ohraničená tagem `Wizard`. Určuje se zde, kolik stránek bude mít průvodce, v našem případě jen jednu, a jaké bude obsahovat editační prvky. Mezi jednotlivými stránkami lze přecházet pomocí tlačítek, které jsou součástí průvodce. Pokud je vyplněna jen jedna stránka, tlačítka jsou neaktivní.

MujProjektSolution.vstemplate*

```
<VSTemplate
  Version="2.0"
  Type="ProjectGroup"
  xmlns="http://schemas.microsoft.com/developer/vstemplate/2005">
  <TemplateData>
    <Name>UkazkaPouzitiGP MujProjekt Solution</Name>
    <Description>UkazkaPouzitiGP MujProjekt Solution</Description>
    <ProjectType>CSharp</ProjectType>
    <SortOrder>90</SortOrder>
    <Icon>MujProjektSolution.ico</Icon>
    <CreateNewFolder>false</CreateNewFolder>
    <DefaultName>UkazkaPouzitiGPSolution</DefaultName>
    <ProvideDefaultName>true</ProvideDefaultName>
  </TemplateData>
  <TemplateContent>
    <ProjectCollection>
      <ProjectTemplateLink ProjectName="$JmenoProjektu$">
        Projects\MujProjekt\MyTemplate.vstemplate</ProjectTemplateLink>
    </ProjectCollection>
  </TemplateContent>
  <WizardExtension>
    <Assembly>Microsoft.Practices.RecipeFramework.VisualStudio,
      Version=2.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35</Assembly>
    <FullClassName>Microsoft.Practices.RecipeFramework.VisualStudio.Templates.UnfoldTemplate</FullClassName>
  </WizardExtension>
  <WizardData>
    <Template xmlns="http://schemas.microsoft.com/pag/gax-template"
              SchemaVersion="1.0"
              Recipe="CreateSolution">
    </Template>
  </WizardData>
</VSTemplate>
```

100 %

Obrázek 63: Obsah souboru reprezentující „Solution“

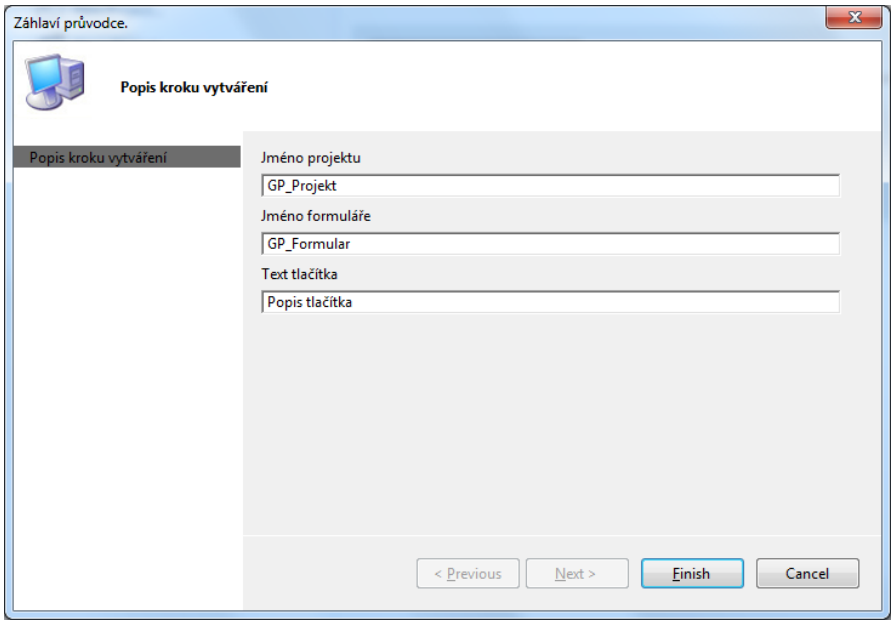
UkazkaPouzitiGP.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<GuidancePackage xmlns="http://schemas.microsoft.com/pag/gax-core"
    Name="UkazkaPouzitiGP"
    Caption="UkazkaPouzitiGP"
    Description="Ukazka pouziti Guidance Package pro potrebu diplomove prace"
    Guid="34cc9860-ead3-4e3d-baa6-bc37c89e1087"
    SchemaVersion="1.0">
  <Recipes>
    <Recipe Name="CreateSolution">
      <Caption>Záhlaví průvodce.</Caption>
      <Arguments>
        <Argument Name="JmenoProjektu">
          <Converter Type="Microsoft.Practices.RecipeFramework.Library.Converters.NamespaceStringConverter,
                  Microsoft.Practices.RecipeFramework.Library"/>
        </Argument>
        <Argument Name="JmenoForm">
          <Converter Type="Microsoft.Practices.RecipeFramework.Library.Converters.NamespaceStringConverter,
                  Microsoft.Practices.RecipeFramework.Library"/>
        </Argument>
        <Argument Name="TextTlacitko" Type="System.String"/>
      </Arguments>
      <GatheringServiceData>
        <Wizard xmlns="http://schemas.microsoft.com/pag/gax-wizards" SchemaVersion="1.0">
          <Pages>
            <Page>
              <Title>Popis kroku vytváření</Title>
              <Fields>
                <Field Label="Jméno projektu" ValueName="JmenoProjektu" />
                <Field Label="Jméno formuláře" ValueName="JmenoForm" />
                <Field Label="Text tlačítka" ValueName="TextTlacitko" />
              </Fields>
            </Page>
          </Pages>
        </Wizard>
      </GatheringServiceData>
    </Recipe>
  </Recipes>
</GuidancePackage>
```

100 %

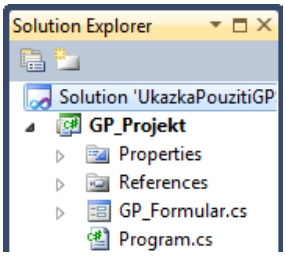
Obrázek 64: Obsah konfiguračního souboru Guidance Package

Jakmile máme toto vše připravené, můžeme spustit náš první vlastní Guidance Package. Spuštění provedeme ve vývojovém modu Visual Studia stisknutím klávesové zkratky „Ctrl + F5“ nebo přes menu „Debug > Start Without Debugging“. Jakmile se spustí VS a vybereme vytvořit nový projekt, v dialogu „New Project“ uvidíme náš GP a po jeho označení, pojmenování a potvrzení se zobrazí dialog (viz. Obrázek 65). Vidíme tři textová pole, která je třeba vyplnit, a po potvrzení již máme založený nový projekt.

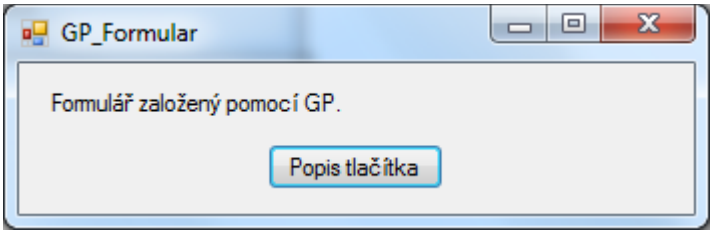


Obrázek 65: Dialog nastavující vlastnosti ukázkového projektu GP

O tom, že nastavení jména projektu a formuláře bylo akceptováno, se můžeme přesvědčit v panelu „Solution Explorer“ (viz. Obrázek 66) a o vyplnění textu tlačítka se můžeme přesvědčit při spuštění projektu (viz. Obrázek 67) nebo v desigeneru formuláře.



Obrázek 66: Nově založený projekt z ukázkového GP



Obrázek 67: Formulář s popiskem na tlačítku, který byl zadán v průvodci vytvoření GP

6.5 Nasazení

Nasazení GP do vývojového prostředí se provádí přes soubor, který se vytvoří při sestavování balíčku. Tímto souborem je soubor s příponou *.vsix. Standardně je tento soubor umístěn ve složce „<*projekt*>/bin/Release“. Odinstalování balíčku se provádí přes dialog „Extension Manager“ ve VS, nalezneme jej v menu „Tools/Extension Manager …“.

Při vytvoření nového projektu za pomoci Guidance Package se automaticky vytvoří soubor „<*solutionName*>.gpState“, který obsahuje jméno použitého GP a jeho verzi. Soubor slouží k tomu, aby se při opětovném otevření projektu načetli do VS i příslušné recepty. Pokud se při vývoji používá nějaký „source control“, musí být na něho nahrán i tento soubor, aby i ostatní vývojáři měli ve VS integrovaný GP, pokud ho mají nainstalovaný ve VS. Pokud se soubor ztratí nebo jeho obsah bude smazán, již nebude možné používat GP při práci s tímto projektem.

7 Závěr

Cílem práce bylo nastudovat a zhodnotit možnosti použití technologií pro automatické generování kódu. Těmito technologiemi byly Code Snippet, Visual Studio Template, T4Template, Domain Specific Language a Guidance Package. Dalším mým úkolem bylo vytvořit ukázky použití jednotlivých technologií. Tato práce by měla sloužit jako úvod do problematiky automatického generování kódu. Některé technologie jsou zde popsány téměř celé a jiné jen stručně, neboť jejich obsáhlost překračuje rozsah této práce.

Code Snippets považuji za první krok k využívání automatického generování kódu. Code Snippets je možné začít používat v jakémkoliv stavu vyvíjeného produktu, od počáteční implementace až po údržbu. Generovat lze však maximálně obsah jednoho souboru, ale to bych příliš nedoporučoval, a na vytváření nového souboru použil VS Template. Minimální velikost generovaného kódu není omezena, omezující je pouze efektivita používání velmi krátkého Code Snippets. Jednoduché vytvoření a používání, které je popsané v kapitole 2, je velkou výhodou této technologie. Možná její snadné použití a implementace je důvodem, proč existuje tak málo materiálů o této problematice a spíš jsou k nalezení hotová řešení.

VS Template by se dalo považovat za pokračování Code Snippets, protože po stránce rozsahu jejich využití začíná tam, kde končí využití Code Snippets. To znamená, že je možno automaticky generovat jeden soubor (pomocí „Item template“), ale i celý projekt (pomocí „Project template“), nebo skupinu projektů (pomocí „ProjectGroup template“). Jako nevýhodu bych považoval komplikovanější tvorbu průvodce pro vstup dat od uživatele. O VS šablonách existuje již více materiálů než ke Code Snippets, a to jak v podobě návodů na internetu, video návodů, tak i v publikacích, které se zabývají problematikou efektivního využívání Visual Studia.

Použití T4 šablon je komplikovanější než u předchozích technologií, z pohledu nalezení správného uplatnění této technologie. Ale s příchodem VS 2010 je možné vytvářet T4 šablony i za běhu aplikace a tímto způsobem generovat výstupy z aplikace pro uživatele. Takovýto způsob použití T4 šablon popsal například Tomáš Herceg, viz [8]. Možností použít T4 šablonu za běhu aplikace se značně zvýšila jejich použitelnost. Výhodou T4 je i stálá aktuálnost generovaného souboru nebo skupiny souborů. Jakmile se změní vstupní data pro T4 šablonu, při příští kompilaci nebo uložení projektu se soubor automaticky přegeneruje. Pokud generujeme z T4 šablony například kód v jazyce C# a chceme jej ještě nějakým způsobem

modifikovat, musíme vytvořit nový soubor a v něm definovat parcial třídu k vygenerované třídě. Velké množství informací lze čerpat z blogu, jehož autorem je Oleg Sych, viz. [7].

DSL je dle mého mínění nejkomplikovanější technologií, která je v této práci zmíněna, neboť vyžaduje velkou míru abstrakce, protože je nutná kompletní znalost problému, pro který budeme abstrakci vymýšlet a vytvářet. Aby DSL mělo nějaký dále použitelný výstup, než jen grafické znázornění, například v podobě tříd nebo třeba SQL skriptu, je zapotřebí použít T4 šablony pro jejich generování. Nejznámější použití DSL mezi .NET vývojáři jsou nástroje pro vytváření datových modelů aplikací, jako jsou DataSety, LINQ to SQL a EntityFramework. O doménové specifikaci existuje několik publikací, které jsou více či méně obecné. Pro pochopení konkrétní implementace DSL ve VS jsem použil knihu, viz [12].

Guidance Package považuji za poslední krok k integraci technologií pro automatické generování kódu do VS. Touto technologií je možno všechny předchozí integrovat do jednoho balíku, a tímto způsobem vytvořit ucelený blok, který bude generovat všechny fragmenty pro konkrétní typ projektu. Nevýhodou je nutnost mít u solution souboru soubor *.gpState a také fakt, že Microsoft přestal tuto technologii dále vyvíjet, ale ve VS 2010 je zatím stále podporována. GP by mělo být nahrazeno takzvanými „Extension“ (rozšíření), ale tam vidím problém v tom, že pokud těchto rozšíření bude ve VS více, budou vidět všechny v každém druhu projektu. K seznámení s problematikou GP lze použít seriál návodů od Jelle Druyts, viz. [13], kde je sice tento tutoriál psaný pro VS 2005, ale je stále použitelný i pro VS 2010. Pro hlubší seznámení s problematikou lze potom použít stránky MSDN, viz [14].

Během psaní této práce a uplatňování načerpaných informací v praxi jsem pochopil, že čas investovaný do studia a tvorby generátorů kódu se několikanásobně v budoucnu vrátí. Bohužel, jak jsem v praxi zjistil, jen málo vývojářů se o automatické generování kódu zajímá a aktivně ho používá.

Pro vývojáře, kteří mají zájem se s automatickým generováním kódu seznámit, považuji tuto práci spolu s ukázkovými kódy za plně dostačující. Pro ty, kteří uvažují o hlubším studiu DSL nebo i jiné technologie, je tato práce dobrým začátkem pro další prohlubování znalostí a odkazy v seznamu použité literatury mohou posloužit jako rozcestník k nejrůznějším informacím.

8 Literatura

[1] MSDN. *Code Snippets Schema Reference* [online]. [cit. 2009-05-20]. Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/ms171418(VS.80).aspx>

[2] MSDN. *Visual Studio 2005 Code Snippets* [online]. [cit. 2009-05-20]. Dostupné z WWW: < http://msdn.microsoft.com/en-us/vstudio/aa718338.aspx>

[3] MSDN. *Visual Studio Template Schema Reference* [online]. [cit. 2009-05-20]. Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/xwkxbww4(VS.80).aspx>

[4] MSDN. *Visual Studio Templates* [online]. [cit. 2009-05-20]. Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/6db0hwky(VS.80).aspx>

[5] MILES, Dylan. *Create Custom Project and Item Templates for Visual Studio 2005?* [online]. [cit. 2009-05-20]. Dostupné z WWW: <http://msdn.microsoft.com/en-us/vstudio/video/Bb870452>

[6] NAYYERI, K. *Wrox Professional Visual Studio Extensibility*. March, 2008. ISBN: 978-0-470-23084-8.

[7] SYCH, Oleg. *Text Template Transromation toolkit* [online]. 22. 12. 2007 [cit. 2010-01-20]. Dostupné z WWW: <http://www.olegsych.com/2007/12/text-template-transformation-toolkit/>

[8] HERCEG, Tomáš. *Používáme T4 šablony* [online]. 27. 5. 2010 [cit. 2010-06-20]. Dostupné z WWW: <http://www.vbnet.cz/clanek--158-pouzivame_t4_sablony.aspx>

[9] MSDN. *Code Generation and T4 Text Templates* [online]. 22. 12. 2007 [cit. 2010-01-20]. Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/bb126445.aspx>

[10] KOLMAN Daniel. *Model Driven Development v praxi* [online]. 14. 03. 2009 [cit. 2010-05-03]. Dostupné z WWW: <http://dkolman.blogspot.com/2009/03/model-driven-development-v-praxi.html>

[11] WILLS Alan Cameron. *Domain-SpecificVisualization and Modeling SDK (DSL Tools) - Intro Lab*[online]. duben 2010 [cit. 2010-08-03]. Dostupné z WWW: <http://code.msdn.microsoft.com/DSLToolsLab>

[12] COOK, Steve; JONES, Gareth; KENT, Stuart; WILLS, Alan Cameron. *Domain-Specific* Development with Visual Studio DSL Tool.May, 2007. ISBN: 978-0-321-39820-8.

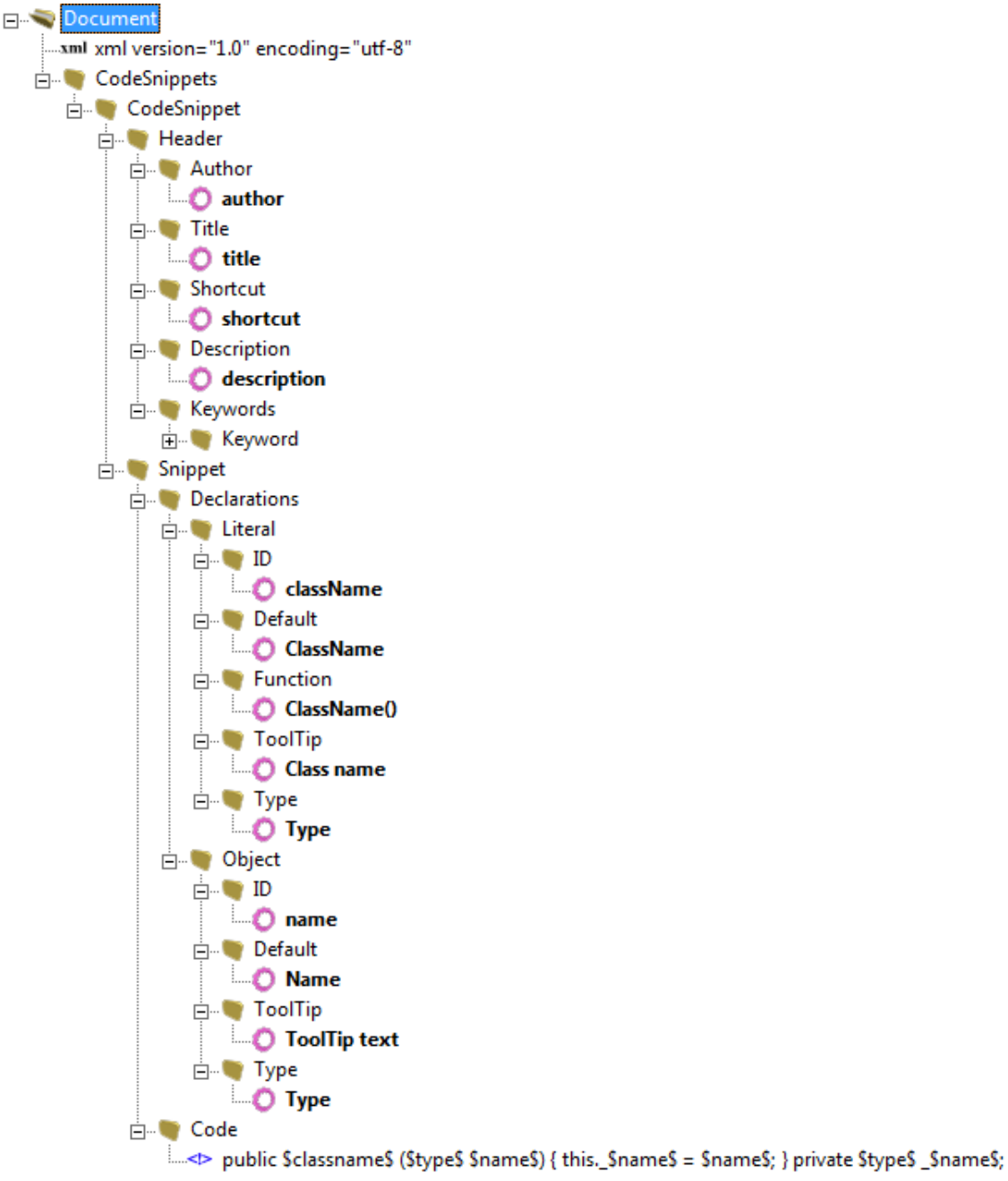
[13] DRUYTS, Jelle. *Guidance Automation* [online]. 2006 [cit. 2011-01-10]. Dostupné z WWW: <http://jelle.druyts.net/CategoryView.aspx?category=Blog|Programming|.NET|GuidanceAutomation>

[14] MSDN. *Developing a Guidance Package* [online]. [cit. 2011-01-10]. Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/ff697248.aspx>

[15] DRUYTS, Jelle. *A Deep Dive Into The Guidance Automation Toolkit (Or: Sit Back And Make It Do Your Work)* [online]. [cit. 2011-01-10]. Dostupné z WWW: <http://www.microsoft.com/belux/msdn/nl/chopsticks/default.aspx?id=10>

Přílohy

A Schéma Code Snippets



Obrázek 68: Schéma Code Snippets (složky jsou XML elementy a kolečka jsou hodnoty elementu)

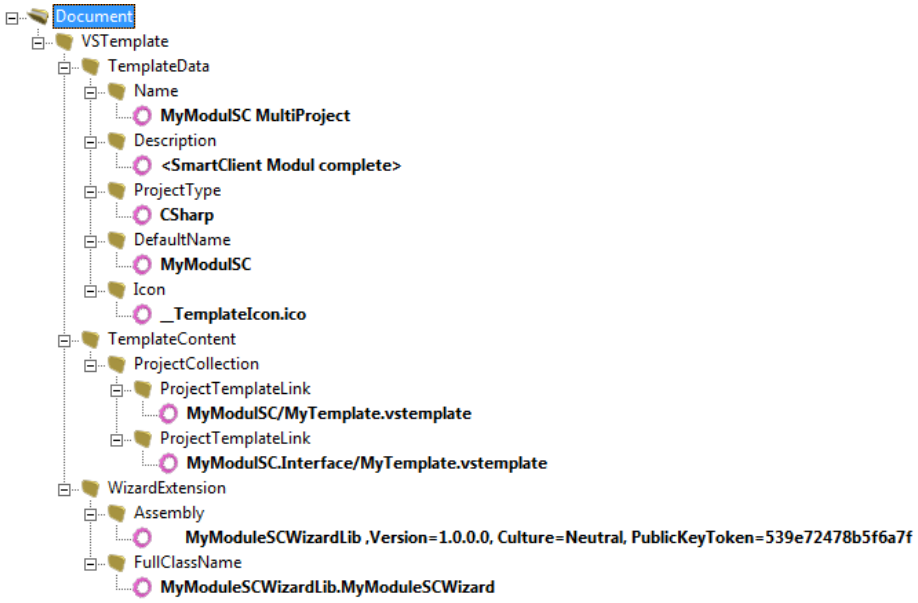
Class.snippet

```
<?xml version="1.0" encoding="utf-8"?>
<CodeSnippets xmlns="http://schemas.microsoft.com/VisualStudio/2005/CodeSnippet">
  <CodeSnippet Format="1.0.0">
    <Header>
      <Author>author</Author>
      <Title>title</Title>
      <Shortcut>shortcut</Shortcut>
      <Description>description</Description>
      <Keywords>
        <Keyword>key word</Keyword>
      </Keywords>
    </Header>
    <Snippet>
      <Declarations>
        <Literal>
          <ID>className</ID>
          <Default>ClassName</Default>
          <Function>ClassName()</Function>
          <ToolTip>Class name</ToolTip>
          <Type>Type</Type>
        </Literal>
        <Object>
          <ID>name</ID>
          <Default>Name</Default>
          <ToolTip>ToolTip text</ToolTip>
          <Type>Type</Type>
        </Object>
      </Declarations>
      <Code Language="CSharp">
        <![CDATA[public $classname$ ($type$ $name$)
                    {
                        this._$name$ = $name$;
                    }
                    private $type$ _$name$;
]]>
      </Code>
    </Snippet>
  </CodeSnippet>
</CodeSnippets>
```

100 %

Obrázek 69: Kód Code Snippets

B Schéma Visual Studio Template



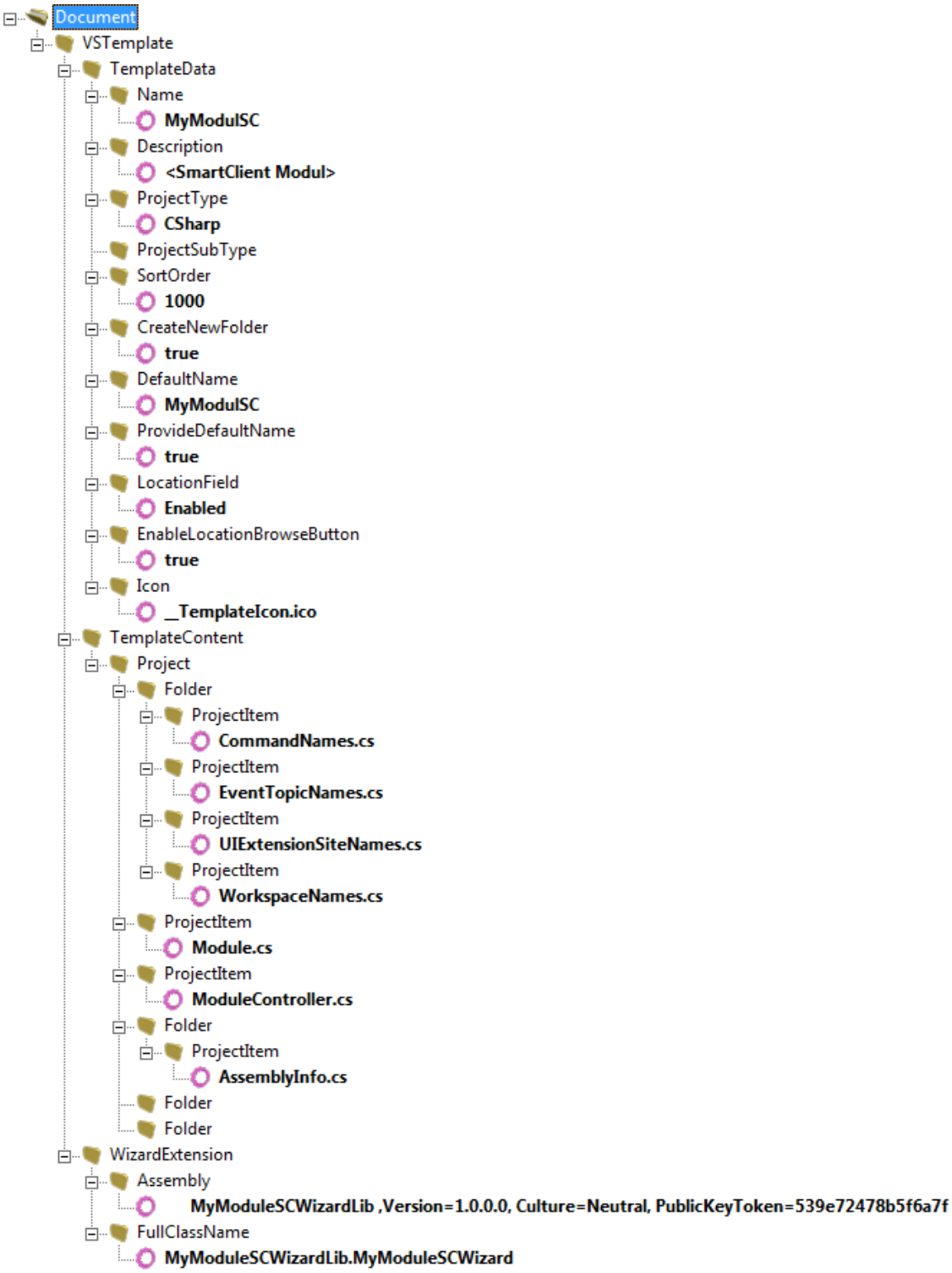
Obrázek 70: Schéma šablony MultiProjektu

MyTemplate_Multi.vstemplate

```
1  <VSTemplate
2    Version="2.0.0"
3    xmlns="http://schemas.microsoft.com/developer/vstemplate/2005"
4    Type="ProjectGroup">
5    <TemplateData>
6      <Name>MyModulSC MultiProject</Name>
7      <Description>&lt;SmartClient Modul complete&gt;</Description>
8      <ProjectType>CSharp</ProjectType>
9      <DefaultName>MyModulSC</DefaultName>
10     <Icon>__TemplateIcon.ico</Icon>
11   </TemplateData>
12   <TemplateContent>
13     <ProjectCollection>
14       <ProjectTemplateLink ProjectName="MyModulSC">
15         MyModulSC/MyTemplate.vstemplate
16       </ProjectTemplateLink>
17       <ProjectTemplateLink ProjectName="MyModulSC.Interface">
18         MyModulSC.Interface/MyTemplate.vstemplate
19       </ProjectTemplateLink>
20     </ProjectCollection>
21   </TemplateContent>
22   <WizardExtension>
23     <Assembly>
24       MyModuleSCWizardLib ,Version=1.0.0.0, Culture=Neutral, PublicKeyToken=539e72478b5f6a7f
25     </Assembly>
26     <FullClassName>MyModuleSCWizardLib.MyModuleSCWizard</FullClassName>
27   </WizardExtension>
28 </VSTemplate>
```

100 %

Obrázek 71: Kód šablony MultiProjektu



Obrázek 72: Schéma šablony projektu MyModulSC/MyTemplate.vstemplate z předchozího MultiProjektu

MyTemplate_Modul.vstemplate

```
<VSTemplate Version="2.0.0" xmlns="http://schemas.microsoft.com/developer/vstemplate/2005" Type="Project">
  <TemplateData>
    <Name>MyModulSC</Name>
    <Description>&lt;SmartClient Modul&gt;</Description>
    <ProjectType>CSharp</ProjectType>
    <ProjectSubType>
    </ProjectSubType>
    <SortOrder>1000</SortOrder>
    <CreateNewFolder>true</CreateNewFolder>
    <DefaultName>MyModulSC</DefaultName>
    <ProvideDefaultName>true</ProvideDefaultName>
    <LocationField>Enabled</LocationField>
    <EnableLocationBrowseButton>true</EnableLocationBrowseButton>
    <Icon>__TemplateIcon.ico</Icon>
  </TemplateData>
  <TemplateContent>
    <Project TargetFileName="$ModulName$.csproj" File="MyModulSC.csproj.pok" ReplaceParameters="true">
      <Folder Name="Constants" TargetFolderName="Constants">
        <ProjectItem ReplaceParameters="true" TargetFileName="CommandNames.cs">CommandNames.cs</ProjectItem>
        <ProjectItem ReplaceParameters="true" TargetFileName="EventTopicNames.cs">EventTopicNames.cs</ProjectItem>
        <ProjectItem ReplaceParameters="true" TargetFileName="UIExtensionSiteNames.cs">UIExtensionSiteNames.cs</ProjectItem>
        <ProjectItem ReplaceParameters="true" TargetFileName="WorkspaceNames.cs">WorkspaceNames.cs</ProjectItem>
      </Folder>
      <ProjectItem ReplaceParameters="true" TargetFileName="Module.cs">Module.cs</ProjectItem>
      <ProjectItem ReplaceParameters="true" TargetFileName="ModuleController.cs">ModuleController.cs</ProjectItem>
      <Folder Name="Properties" TargetFolderName="Properties">
        <ProjectItem ReplaceParameters="true" TargetFileName="AssemblyInfo.cs">AssemblyInfo.cs</ProjectItem>
      </Folder>
      <Folder Name="Services" TargetFolderName="Services" />
      <Folder Name="Views" TargetFolderName="Views" />
    </Project>
  </TemplateContent>
  <WizardExtension>
    <Assembly>
      MyModuleSCWizardLib ,Version=1.0.0.0, Culture=Neutral, PublicKeyToken=539e72478b5f6a7f
    </Assembly>
    <FullClassName>MyModuleSCWizardLib.MyModuleSCWizard</FullClassName>
  </WizardExtension>
</VSTemplate>
```

100 %

Obrázek 73: Kód šablony projektu MyModulSC/MyTemplate.vstemplate z předchozího MultiProjektu

C Přehled proměnných ve Visual Studio Templates

Parameter	Description
clrversion	Current version of the common language runtime (CLR).
GUID [1-10]	A GUID used to replace the project GUID in a project file. You can specify up to 10 unique GUIDs (for example, guid1).
itemname	The name provided by the user in the Add New Item dialog box.
machinename	The current computer name (for example, Computer01).
projectname	The name provided by the user in the New Project dialog box.
registeredorganization	The registry key value from HKLM\Software\Microsoft\Windows NT\CurrentVersion\RegisteredOrganization.
rootnamespace	The root namespace of the current project. This parameter is used to replace the namespace in an item being added to a project.
safeitemname	The name provided by the user in the Add New Item dialog box, with all unsafe characters and spaces removed.
safeprojectname	The name provided by the user in the New Project dialog box, with all unsafe characters and spaces removed.
time	The current time in the format DD/MM/YYYY 00:00:00.
userdomain	The current user domain.
username	The current user name.
year	The current year in the format YYYY.

D Obsah přiloženého CD

Struktura CD:

- Zdrojové kódy – ukázky kódů jednotlivých technologií
  - Code Snippets
  - VS Template
  - T4
  - DSL
  - Guidance Package
- Text – text Diplomové práce